REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DES FRERES MENTOURI CONSTANTINE
FACULTE DES SCIENCES DE LA TECHNOLOGIE
DEPARTEMENT D'ELECTRONIQUE

THESE

Présentée pour obtenir le diplôme de

DOCTORAT EN SCIENCES

Spécialité : ELECTRONIQUE
Option : CONTROLE DES SYSTEMES

Par :

MERABTI Halim

THEME

COMMANDE PREDICTIVE PAR LA THEORIE DES INTERVALLES FLOUS ET METAHEURISTIQUES

Devant le Jury :

Président	Z. HAMMOUDI	*Professeur, Université de Constantine*
Rapporteur	K. BELARBI	*Professeur, Ecole polytechnique de Constantine*
Examinateurs	A. FARROUKI	*Professeur, Université de Constantine*
	N. SLIMANE	*Maitre de conférences A, Université de Batna*
	S. BOUOUDEN	*Maitre de conférences A, Université de Khenchela*

Année : 2015

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DES FRERES MENTOURI CONSTANTINE
FACULTE DES SCIENCES DE LA TECHNOLOGIE
DEPARTEMENT D'ELECTRONIQUE

THESE

Présentée pour obtenir le diplôme de

DOCTORAT EN SCIENCES

Spécialité : ELECTRONIQUE
Option : CONTROLE DES SYSTEMES

Par :

MERABTI Halim

THEME

COMMANDE PREDICTIVE PAR LA THEORIE DES INTERVALLES FLOUS ET METAHEURISTIQUES

Devant le Jury :

Président	Z. HAMMOUDI	*Professeur, Université de Constantine*
Rapporteur	K. BELARBI	*Professeur, Ecole polytechnique de Constantine*
Examinateurs	A. FARROUKI	*Professeur, Université de Constantine*
	N. SLIMANE	*Maitre de conférences A, Université de Batna*
	S. BOUOUDEN	*Maitre de conférences A, Université de Khenchela*

Année : 2015

Dédicaces

A mes parents,

A ma petite famille,

A mes frères et sœurs,

A tous mes amis,

Remerciements

Avant tout, mes remerciements à **ALLAH** le tout-puissant qui m'a aidé à faire ce travail et qui m'a donné le courage, la patience et la santé durant toutes ces longues années d'études afin que je puisse arriver à ce niveau.

Ce travail de recherche a été effectué à l'Université de Constantine, sous la direction scientifique de Monsieur le professeur khaled BELARBI, que je tiens à remercier très vivement de son enthousiasme envers mon travail, de sa disponibilité et de son soutien scientifique et humain. Je le remercie, également, pour la confiance qu'il m'a accordée, ses encouragements et ses précieux conseils.

Je suis très honoré que Monsieur Zoheir HAMMOUDI**,** professeur à l'université de Constantine, ait accepté de présider le jury de cette thèse. Je remercie également Messieurs Atef FARROUKI, Professeur à l'université de Constantine, Noureddine SLIMANE, Maître de Conférence "A" à l'Université de Batna et Sofian BOUOUDEN, Maître de Conférence "A" à l'Université de Khenchela, d'avoir accepté de participer à ce jury.

Mes remerciements s'adressent également à tous mes collègues de l'Unité de Recherche en Matériaux Avances de Annaba, leurs soutiens et particulièrement le directeur de l'Unité Monsieur H. MERADI, pour son aide et sa flexibilité qui m'a donné la possibilité d'avoir un compromis entre les travaux de recherche au niveau de l'unité et mes travaux de thèse.

Table des matières

Remerciements i

Liste des figures v

Liste des tableaux viii

Liste d'acronymes ix

Introduction Générale 1

Chapitre 1 : La commande prédictive 4

1.1. Introduction 4

1.2. Principe de la commande prédictive 4

1.3. Modèles de prédiction 5

1.3.1. Les modèles de prédictions linéaires 5

1.3.2. Les modèles de prédictions non-linéaires 6

1.4. Méthode d'optimisation 7

1.5. Méthode d'application de la commande 7

1.6. La commande prédictive linéaire sans contraintes 8

1.6.1. Calcul des prédictions 8

1.6.2. Calcul de la commande 10

1.7. La commande prédictive linéaire avec contraintes 11

1.7.1. Méthodes de solutions de la commande prédictive linéaire avec contraintes 12

1.7.1.1. Méthode de l'ensemble actif 12

1.7.1.2. Point intérieur 12

1.7.1.3. Programmation Quadratique multiparamétrique (MPQP) 13

1.7.1.4. Commande prédictive approximée 13

1.8. La commande prédictive non-linéaire 13

1.8.1. Formulation générale de la NMPC 13

1.8.2. Solution du problème de la commande prédictive non-linéaire 14

1.9. Stabilité de la commande prédictive 14

1.10. Conclusion 15

Chapitre 2 : Commande prédictive robuste par les intervalles flous 16
2.1. Introduction 16
2.2. Concepts pertinents et notations 16
2.2.1. Définition 1: Intervalle conventionnel 16
2.2.2. Propriétés algébriques principales 17
2.2.3. Non-existence des éléments inverses 17
2.2.4. Définition 2: les ensembles flous 18
2.2.5. Définition 3: L'alfa coupe 18
2.2.6. Définition 4: Les intervalles flous 18
2.3. Modèle d'état par les intervalles flous 20
2.3.1. Propriété 1 21
2.3.2. Démonstration 21
2.4. MPC Robuste 23
2.5. Résultats de simulation 24
2.5.1. Exemple 1 24
2.5.2. Exemple 2 28
2.6. Conclusion 31

Chapitre 3 : Solution de la commande prédictive par les métaheuristiques 32
3.1. Introduction 32
3.2. Optimisation par essaim de particules (PSO) 32
3.3. Optimisation par colonie de fourmis (ACO) 35
3.4. Optimisation par Algorithme de recherche gravitationnel (GSA) 39
3.5. Application 41
3.6. Résultats de simulation 42
3.6.1. Système linéaire 42
3.6.1.1. Système mono variable 42
3.6.1.2. Système multi variables 44
3.6.2. Système non-linéaire 47
3.7. Résultats expérimentaux 57
3.7.1. Le robot LEGO 57
3.7.2. Modèle cinématique du robot mobile utilisé 58
3.7.3. Commande du robot LEGO 58

3.7.3.1. Scénario 01 : suivi d'une trajectoire circulaire 58
3.7.3.2. Scénario 02 : suivi d'une trajectoire rectiligne et évitement d'obstacles 61
3.8. Conclusion 63

Chapitre 4 : La commande prédictive multi-objectifs

4.1. Introduction 64
4.2. L'optimisation multi-objectifs 64
4.3. Métaheuristiques pour l'optimisation multi-objectifs 65
4.3.1. Programmation de l'algorithme génétique multi-objectifs NSGAII 66
4.3.2. Algorithme de l'essaim de particules pour l'optimisation multi-objectifs 68
4.3.2.1. Principe du MOPSO 68
4.3.2.2. Principe du NSPSO 69
4.3.3. Optimisation multi-objectifs par Algorithme de recherche gravitationnel 70
4.4. La commande prédictive multi-objectifs 71
4.5. Résultats de simulation 72
4.5.1. Exemple1 72
4.5.2. Exemple2 75
4.6. Conclusion 78

Conclusion Générale 79

Références 81

Liste des figures

1.1. Schéma de principe de la commande prédictive ..... …4
2.1. Représentation d'un intervalle flou « trapézoïdal » ..... …19
2.2. Schéma de base de l'algorithme d'identification ..... …20
2.3. Schéma de la commande prédictive robuste ..... …23
2.4.a. Intervalle flou de R ..... …24
2.4.b. Intervalle flou de L ..... …24
2.5. Perturbation sur la charge ..... …25
2.6. La vitesse angulaire dans le cas de la MBPC simple ..... …26
2.7. Signal de commande dans le cas de la MBPC simple ..... …26
2.8. Vitesse angulaire dans le cas d'une MBPC robuste avec une variation dans R, L et une perturbation sur la charge ..... …27
2.9. Signal de commande dans le cas de la MBPC robuste ..... …27
2.10.a. Intervalle flou de a_2 ..... …28
2.10.b. Intervalle flou de a_3 ..... …28
2.11.Concentration dans le cas de MBPC non robuste avec des variations sur les paramètres29
2.12. Signal de commande dans le cas de MBPC non robuste ..... …30
2.13. Concentration dans le cas de MBPC robuste ..... …30
2.14. Signal de commande dans le cas de MBPC robuste ..... …31
3.1. Schéma de principe du déplacement d'une particule ..... …33
3.2. Expérience pour la sélection du chemin le plus court ..... …35
3.3. a) Distribution discrète de probabilité b) Fonction de densité de probabilité(FDP) .... …37
3.4. Structure de l'archive des solutions ..... …38
3.5. Signal de commande pour l'exemple1, PSO, ACO et GSA ..... …43
3.6. Etats pour exemple1, PSO, ACO et GSA ..... …43
3.7. Le signal de commande pour l'exemple 2, PSO et ACO ..... …45
3.8. Etats pour exemple2, PSO et ACO ..... …46
3.9. Le signal de commande pour l'exemple 2, GSA ..... …46
3.10. Etats pour exemple2, GSA ..... …47
3.11. Robot mobile utilisé ..... …47

3.12. Trajectoire du robot : MBPC avec PSO et ACO ..... …49
3.13. Les signaux de commande:MBPC avec le PSO et ACO ..... …50
3.14. Trajectoire du robot : MBPC avec GSA ..... …50
3.15. Les signaux de commande:MBPC avec le GSA ..... …51
3.16. Trajectoire du robot pour l'évitement d'obstacles fixes: MBPC avec PSO ..... …52
3.17. Les signaux de commande (MBPC avec PSO) ..... …52
3.18. Trajectoire du robot pour l'évitement d'obstacles : MBPC avec ACO ..... …53
3.19. Les signaux de commande (MBPC avec ACO_R) ..... …53
3.20. Trajectoire du robot pour l'évitement d'obstacles : MBPC avec GSA ..... …54
3.21. Les signaux de commande (MBPC avec GSA) ..... …54
3.22. Résultats du troisième scénario: La trajectoire du robot et les obstacles mobiles .... …55
3.23. Les signaux de commande ..... …56
3.24. La brique Mindstorms NXT ..... …57
3.25. Servomoteur Mindstorms NXT ..... …58
3.26. Le robot Lego Mindstorms NXT ..... …59
3.27.a. Trajectoire du robot ..... …59
3.27.b. Vitesse de la roue droite ..... …60
3.27.c. Signal de commande de la roue droite ..... …60
3.27.d. Vitesse de la roue gauche ..... …60
3.27.e. Signal de commande de la roue gauche ..... …60
3.27.f. Le LEGO entraine de tracer la trajectoire ..... …61
3.28.a. Trajectoire du robot ..... …61
3.28.b. Vitesse de la roue droite ..... …62
3.28.c. Signal de commande de la roue droite ..... …62
3.28.d. Vitesse de la roue gauche ..... …62
3.28.e. Signal de commande de la roue gauche ..... …62
3.28.f. Le robot entraine de tracer la trajectoire ..... …63
3.28.g. Trajectoire tracée par le robot ..... …63
4.1. Principe de l'algorithme NSGAII ..... …67
4.2. Signaux de commande ..... …73
4.3. Etats du système ..... …74
4.4. Trajectoire du robot1 ..... …76
4.5. Signaux de commande pour le robot1..... …76

4.6. Trajectoire du robot2 ..... …77
4.7. Signaux de commande pour le robot1 ..... …77

Liste des tableaux

3.1. Temps de calcul ..... …44
3.2. Temps de calcul pour l'hélicoptère ..... …47
3.3. Paramètres des métaheuristiques ..... …49
3.4. Temps de calcul pour le 1^{ier} scénario ..... …51
3.5. Temps de calcul pour le 2^{iem} scénario ..... …55
3.6. Temps de calcul pour le 3^{iem} scénario ..... …56
4.1. Temps de calcul pour les deux algorithmes ..... …74
4.2. Temps de calcul pour l'exemple 2 ..... …78

Liste d'acronymes:

MPC : Commande prédictive à base de modèle (Model Based Predictive Control)

NMPC : Commande prédictive non-linéaire (Non linear Model Mredictiv Control)

APC : Commande prédictive approximée (Approximate Predective Contro)

MOMPC : Commande prédictive multi-objectifs

QP : Programmation quadratique (Quadratic Programming)

SQP : Programme quadratique séquentiel (Sequential Quadratic Program)

MPQP : Programmation quadratique multiparamétrique (Multiparametric Quadratic Programming)

GA : Algorithme génétique (Genetic Algorithm)

PSO : Optimisation par essaim de particules (Particle Swarm Optimization)

ACO : Optimisation par colonie de fourmis (Ant Colony Optimization)

ACO_R : Algorithme de colonie de fourmis pour les domaines continus

GSA : Algorithme de recherche gravitationnel (Gravitational Search Algorithm)

FDP : Fonction de densité de probabilité

MOGA : Algorithme génétique multi-objectifs

NSGA : Algorithme génétique multi-objectifs utilisant le classement des solutions non dominées

MOPSO : Optimisation par essaim de particules multi-objectifs

NSPSO : Optimisation par essaim de particules multi-objectifs utilisant le classement des solutions non dominées

MOGSA : Optimisation par l'algorithme de recherche gravitationnel multi-objectifs

CARMA: «Controlled Auto Regressive Moving Average»

CSTR : «Continuous Stirred Tank Reactor »

Introduction Générale

Introduction générale

La philosophie de la commande prédictive (MPC pour Model Predictive Control) se résume à l'utilisation d'un modèle afin de prédire le comportement du système à commander et choisir la décision la meilleure au sens d'un certain coût tout en prenant en compte les contraintes [4]. Après les années 70, cette méthode s'est développée rapidement que ce soit dans le domaine industriel ou académique. Ainsi, Au cours des dernières 30 années, la MPC a été appliquée avec succès aux procédés industriels complexes tels que la pétrochimie, le génie des procèdes, la papeterie, etc. toute en montrant sa grande capacité pour la résolution des problèmes d'optimisation complexe avec contraintes.

Le paramètre le plus restrictif pour l'applicabilité de la commande prédictive est le temps de calcul qui reste critique notamment dans les domaines où les fréquences d'échantillonnage sont élevées.

La commande prédictive nominale utilise un modèle pour prédire le comportement futur du système assumant un modèle exact. Avec la présence des incertitudes dans les paramètres du système, les performances seront dégradées lorsqu'il y a une différence significative entre le modèle de prédiction et le modèle réel du système. Pour résoudre ce problème, plusieurs stratégies robustes de la commande prédictive tenant en compte les incertitudes sur le modèle ont été proposées [28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]. Cependant, cette amélioration a l'inconvénient d'une augmentation importante dans le temps de calcul qui pourrait devenir moins significative avec le développement des calculateurs [28].

Récemment, les intervalles flous sont devenus un domaine de recherche important parmi la communauté de la logique floue. Par exemple, dans [40], un contrôleur basé sur l'intervalle flou a été conçu et implémenté pour la commande d'un robot mobile. Dans [25] un contrôleur inverse a été développé pour les systèmes linéaires à intervalle flou stable et inversible, dont l'objectif est de maintenir la sortie commandée du système dans une enveloppe de tolérance, autour de la trajectoire désirée, spécifieé par un degré d'appartenance sur la trajectoire floue.

Dans cette thèse, dans la première démarche, nous considérons l'application de la théorie des intervalles flous dans le domaine de l'identification et la commande des systèmes. Le paradigme de base de la commande robuste a été utilisé, c'est-à-dire l'intégration des incertitudes dans la phase de synthèse pour la conception d'un contrôleur prédictif robuste à intervalle flou. La phase identification consistera à construire un modèle

représenté par des intervalles flous en utilisant une base de données. La construction du modèle a été réalisée à l'aide des algorithmes évolutionnaires. Le modèle est alors utilisé pour construire des contrôleurs prédictifs robustes.

Dans la deuxième démarche, nous considérons l'application des métaheuristiques pour la solution du problème d'optimisation de la commande prédictive simple et robuste.

Dans la littérature, plusieurs méthodes ont été proposées pour la résolution du problème d'optimisation de la MPC. L'algorithme le plus simple pour résoudre le problème d'optimisation de la MPC non-linéaire est le programme quadratique séquentiel [7]. D'autres algorithmes ont été proposés pour résoudre ce problème tel que : la méthode de tir multiple, *multiple shooting* [15], somme non-linéaire des carrés *nonlinear sum of squares* [16, 17] et d'autre [18].

Récemment, la communauté scientifique a été orientée vers une nouvelle classe de méthodes, nommées métaheuristiques qui sont des algorithmes proposés pour résoudre des problèmes d'optimisation difficiles. Ces algorithmes peuvent donner assez rapidement une bonne approximation de l'optimum global.

Dans la troisième démarche, une étude sur la capacité des métaheuristiques multi-objectifs de résoudre le problème d'optimisation apparaissant dans la commande prédictive multi-objectifs et la possibilité d'appliquer cette commande pour commander des systèmes présentant une dynamique rapide.

Pour cela, cette thèse est organisée comme suit :

Dans le premier chapitre, le principe de base de la commande prédictive sera présenté dans la première section. La formalisation de la commande prédictive linéaire et non-linéaire sera abordée dans la deuxième et la troisième section, respectivement. La stabilité de la commande prédictive sera discutée dans la quatrième section.

Dans le deuxième chapitre, les intervalles flous sont utilisés pour décrire les incertitudes structurées sur des modèles linéaires. Un contrôleur prédictif robuste a été développé et les résultats de simulation de deux exemples sont présentés.

Le troisième chapitre sera consacré à l'étude de l'applicabilité des métaheuristiques pour la solution du problème d'optimisation de la commande prédictive linéaire et non-linéaire. Les algorithmes suivants seront présentés : l'algorithme d'optimisation par essaim de particules, l'algorithme d'optimisation par colonie de fourmis et l'algorithme de recherche gravitationnel. Ces algorithmes seront utilisés pour résoudre le problème d'optimisation de la

commande prédictive linéaire et non-linéaire. Les résultats de simulations et expérimentaux seront présentés.

Dans le quatrième chapitre, nous considérons la commande prédictive multi-objectifs. Le problème d'optimisation de la commande prédictive multi-objectifs est résolu par les métaheuristiques multi-objectifs. Pour cela, quatre algorithmes seront présentés : l'algorithme génétique pour l'optimisation multi-objectifs (NSGAII), l'algorithme d'optimisation par essaim de particules pour l'optimisation multi-objectifs (MOPSO), l'algorithme d'optimisation par essaim de particules par le classement des solutions non-dominées (NSPSO) et l'algorithme de recherche gravitationnel pour l'optimisation multi-objectifs (MOGSA). Les résultats de simulations seront présentés.

Chapitre 1

La Commande Prédictive

1.1. Introduction:

La commande prédictive à base de modèle, MBPC, est l'une des rares techniques de commande avancée avoir un impact significatif et répandu sur la commande de processus industriels. Elle a été développée et employée dans l'industrie pendant presque vingt ans avant d'attirer l'attention de beaucoup de chercheurs dans le domaine de l'automatique.

Aujourd'hui, vu à l'essor technologique des calculateurs et après l'apparition de méthodes d'optimisation rapides, plusieurs chercheurs se sont focalisés sur l'adaptation de la MPC à la commande des systèmes présentant une dynamiques rapides où les fréquences d'échantillonnages sont très élevées tel que la robotique, l'aérospatial, l'automobile, etc.…

Dans la première section de ce chapitre, nous présentons le principe de base de la commande prédictive, dans la deuxième et la troisième section, nous abordons la formalisation de la commande prédictive linéaire et non-linéaire. La stabilité de la commande prédictive est discutée dans la quatrième section de ce chapitre.

1.2. Principe de la commande prédictive :

L'idée principale de la commande prédictive est basée sur [1]:

- l'utilisation d'un modèle du système à commander pour prédire sa sortie sur un certain horizon
- l'élaboration d'une séquence de commandes futures minimisant une fonction coût
- l'application du premier élément de la séquence optimale précédente sur le système et la répétition de la procédure complète à la prochaine période d'échantillonnage

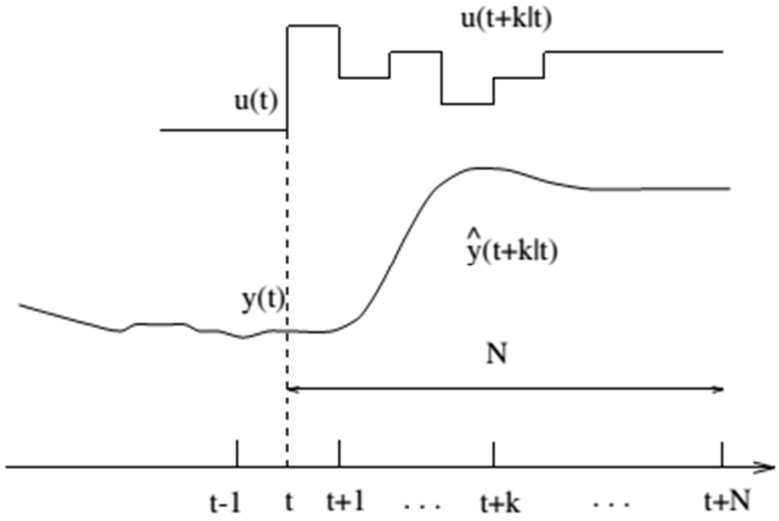


Figure 1.1 : Schéma de principe de la commande prédictive

Donc, la commande prédictive impose, à chaque période d'échantillonnage, une résolution d'un problème d'optimisation.

Le paramètre le plus restrictif pour l'applicabilité de la commande prédictive est le temps de calcul qui reste critique pour les systèmes rapides échantillonnés à haute fréquence. Il est, en effet, nécessaire, pendant une période de temps d'échantillonnage que l'on peut trouver, en fonction de la dynamique finale, une solution admissible au problème d'optimisation. Pour les systèmes lents à grande période d'échantillonnage, l'application des méthodes numériques ne pose pas de problème.

1.3. Modèle de prédiction :

Le modèle du système est un élément important d'une stratégie de commande prédictive. En effet, il doit reproduire avec une exactitude suffisante les caractéristiques dynamiques du système sur un horizon de temps fini. Supposons qu'un système dynamique avec une entrée *u(k)* et une sortie *y(k)*. Le modèle de prédiction qui a pour sortie $\hat{y}(k)$, a pour objectif la prédiction de la sortie du système à des instants futurs (échantillonnés). On distingue deux grandes familles de modèles, linéaires et non-linéaires.

1.3.1. Les modèles de prédictions linéaires :

Plusieurs formes de modèles de prédiction linéaires existent [2], on peut citer ici les modèles à base de:

- Réponse impulsionnelle : les prédictions sont données par la formule

$$\hat{y}(k+j|k) = \sum_{i=1}^{N} h_i . u(k+j-i|k) \quad (1.1)$$

  Où h_i sont les valeurs de la sortie quand le système est excité par une impulsion unitaire. L'avantage de cette méthode est qu'aucune information préalable sur le système n'est nécessaire, de sorte que le procédé d'identification soit simplifié.

- Réponse indicielle: le prédicteur est donné par l'expression suivante :

$$\hat{y}(k+j|k) = \sum_{i=1}^{N} g_i . \Delta u(k+j-i|k) \quad (1.2)$$

  Où g_i sont les valeurs de la sortie quand le système est excité par un échelon unitaire.

- Fonction de transfert : on suppose que le système peut être représenté par un modèle CARMA (Controlled Auto Regressive Moving Average), les prédictions sont données par l'expression :

$$\hat{y}(k+j|k) = \frac{B(z^{-1})}{A(z^{-1})} u(k+j|k) \quad (1.3)$$

Avec :

$$A(z^{-1}) = 1 + a_1.z^{-1} + a_2.z^{-2} + \cdots + a_{n_a}.z^{-n_a} \quad (1.4)$$

$$B(z^{-1}) = 1 + b_1.z^{-1} + b_2.z^{-2} + \cdots + b_{n_b}.z^{-n_b} \quad (1.5)$$

Où :

z^{-1} : représente l'opérateur de retard

B, A: sont respectivement des polynômes associés à l'entrée et à la sortie.

Cette représentation est également valide pour les systèmes instables et présente l'avantage qu'elle a besoin seulement de quelques paramètres, bien que la connaissance a priori du système soit fondamentale, en particulier l'ordre des polynômes *A* et *B*.

- Représentation d'état: supposons qu'on dispose d'un modèle de référence sous la forme suivante :

$$\begin{cases} x(k+1) = A.x(k) + B.u(k) \\ y(k) = C.x(k) \end{cases} \quad (1.6)$$

Où $x$est le vecteur d'état, *u* est l'entré, *y* est la sortie du modèle de référence. *A*, *B* et *C* sont les matrices d'évolution, de commande et d'observation respectivement.

Le modèle de prédiction est donné par :

$$\hat{y}(k+j|k) = C[A^j x(k) + \sum_{i=1}^{j} A^{i-1} B u(k+j-i|k)] \quad (1.7)$$

1.3.2. Les modèles de prédictions non-linéaires :

D'autres formes de modèle peuvent être aussi utilisées, les modèles non-linéaires, qui sont représentés par des équations aux différences non-linéaires, des systèmes à base des réseaux neurones et des systèmes flous. Ces deux dernières techniques sont utilisées en l'absence d'un modèle analytique du système. La forme la plus utilisée pour un modèle de prédiction non-linéaire repose sur une représentation d'état sous la forme suivante:

$$\begin{cases} x(k+1) = f(x(k), u(k)) \\ \hat{y}(k) = h(x(k)) \end{cases} \quad (1.8)$$

Où $f = (f_1, f_2, \ldots, f_n)$est un vecteur de fonctions non-linéaires de forme analytique, et *h* est une fonction analytique non-linéaire.

1.4. Méthode d'optimisation

D'une manière générale, un problème d'optimisation, dans le cas linéaire ou non-linéaire, peut s'écrire sous la forme suivante :

$$min_{\mathcal{U}} J_N(x, k, \mathcal{U}) \tag{1.9}$$

Avec : $$J_{N_p}(x, k, \mathcal{U}) = F\left(x(k + N_p)\right) + \sum_{i=k}^{k+N_p-1} L(x(i), u(i)) \tag{1.10}$$

Où :

- $\sum_{i=k}^{k+N_p-1} L(x(i), u(i))$ est la fonction coût.
- $F(x(k + N_p))$ est le coût sur l'état terminal.

La solution du problème d'optimisation dépend fortement de la nature du modèle et contraintes [3]. Si le modèle est linéaire, en absence de contraintes une solution analytique est obtenue [4] tandis qu'en présence de contraintes le problème est traité comme un problème quadratique pour lequel des algorithmes de solution très efficaces sont disponibles [5, 6]. Cependant, lorsque le modèle est non-linéaire avec ou sans contraintes, le problème d'optimisation devient non convexe et les procédures de solution deviennent plus laborieux et fastidieux [3, 7]. Un certain nombre de méthodes de solutions ayant proposé une amélioration de temps de calcul, mais la complexité de codage des algorithmes a limitée leur applicabilité [6, 7]. Les métaheuristiques, tels que les algorithmes génétiques, optimisation par essaim de particules, optimisation de colonie de fourmis, sont des heuristiques d'usage général qui ont réussi à résoudre des problèmes d'optimisation difficiles dans un temps de calcul raisonnable [8, 9, 10]. Ces heuristiques, qui seront présentées après dans cette thèse, seront utilisées pour résoudre le problème d'optimisation associe à la commande prédictive (linéaire et non-linéaire).

1.5. Méthode d'application de la commande

Après la détermination de la séquence de commande optimale par l'optimisation de la fonction coût, seul le premier échantillon de commande est appliqué au système en temps réel et toute la séquence est recalculée à chaque période d'échantillonnage. Cette méthode repose sur l'exploitation du principe de l'horizon mobile, c'est-à-dire qu'à chaque période d'échantillonnage, la procédure d'optimisation est répétée avec des nouveaux signaux (entrées et sorties). Il est important de préciser que la solution optimale est définie sur l'horizon de commande *Nu* et non pas sur l'horizon de sortie *Np*. Dans ce cas, en dehors de cet horizon

(c'est à-dire, *Np>Nu*) la commande reste constante (la variation de l'incrément de commande est nulle).

1.6. La commande prédictive linéaire sans contraintes:

1.6.1. Calcul des prédictions :

Le modèle de base utilisé pour le calcul des prédictions est un modèle d'état discret sans perturbations donné par:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= C_y x(k) \\ z(k) &= C_z x(k) \end{aligned} \tag{1.11}$$

Où :

- $k \in \mathbb{Z}^+, x(k) \in \mathbb{R}^n$ est le vecteur d'état à l'instant k
- $u(k) \in \mathbb{R}^l$ est le vecteur de commande à l'instant k
- $y(k) \in \mathbb{R}^{m_y}$ est le vecteur des sorties mesurées
- $z(k) \in \mathbb{R}^{m_z}$ est le vecteur des sorties à contrôler
- $A, B, C_y et C_z$ des matrices de dimension correspondante

Supposons que tout le vecteur d'état $x$est mesurable, c à d,

$$\hat{x}(k|k) = x(k) = y(k) \quad (C_y = I) \tag{1.12}$$

Soient N_u : Horizon de commande et N_p l'horizon de prédiction,

Le calcul des sorties prédit $\hat{z}(k+i|k)$ se fait par itération du modèle (1.11).

$$\hat{x}(k+1|k) = Ax(k) + B\hat{u}(k|k) \tag{1.13}$$

$$\hat{x}(k+2|k) = A\hat{x}(k+1|k) + B\hat{u}(k+1|k) \tag{1.14}$$

$$= A^2 x(k) + AB\hat{u}(k|k) + B\hat{u}(k+1|k) \tag{1.15}$$

.

.

.

$$\hat{x}(k+N_p|k) = A\hat{x}(k+N_p-1|k) + B\hat{u}(k+N_p-1|k) \tag{1.16}$$

$$= A^{N_p}x(k) + A^{N_p-1}B\hat{u}(k|k) + \cdots + B\hat{u}(k + N_p - 1|k) \tag{1.17}$$

Rappelons que les entrées peuvent changer seulement aux instants :

$k, k+1, \ldots, k+N_u-1$, et rester constant après.

Après quelques manipulations, nous obtenons le prédicteur sous la forme matricielle :

$$\begin{bmatrix} \hat{x}(k+1|k) \\ \vdots \\ \hat{x}(k+N_u|k) \\ \hat{x}(k+N_u+1|k) \\ \vdots \\ \hat{x}(k+N_p|k) \end{bmatrix} = \begin{bmatrix} A \\ \vdots \\ A^{N_u} \\ A^{N_u+1} \\ \vdots \\ A^{N_p} \end{bmatrix} x(k) + \begin{bmatrix} B \\ \vdots \\ \sum_{i=0}^{N_u-1} A^iB \\ \sum_{i=0}^{N_u} A^iB \\ \vdots \\ \sum_{i=0}^{N_p-1} A^iB \end{bmatrix} u(k-1)$$

$$+ \begin{bmatrix} B & \cdots & 0 \\ AB+B & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \sum_{i=0}^{N_u-1} A^iB & \cdots & B \\ \sum_{i=0}^{N_u} A^iB & \cdots & AB+B \\ \vdots & \vdots & \vdots \\ \sum_{i=0}^{N_p-1} A^iB & \cdots & \sum_{i=0}^{N_p-N_u} A^iB \end{bmatrix} \begin{bmatrix} \Delta\hat{u}(k|k) \\ \vdots \\ \Delta\hat{u}(k+N_u-1) \end{bmatrix} \tag{1.18}$$

Les prédictions de z sont obtenues par :

$$\begin{aligned} \hat{z}(k+1|k) &= C_z x(k+1|k) \\ \hat{z}(k+2|k) &= C_z \hat{x}(k+2|k) \\ &\vdots \\ \hat{z}(k+N_p|k) &= C_z \hat{x}(k+N_p|k) \end{aligned} \tag{1.19}$$

Ou

$$\begin{bmatrix} \hat{z}(k+1|k) \\ \vdots \\ \hat{z}(k+N_p|k) \end{bmatrix} = \begin{bmatrix} C_z & 0 & \cdots & 0 \\ 0 & C_z & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & C_z \end{bmatrix} \begin{bmatrix} \hat{x}(k+1|k) \\ \vdots \\ \hat{x}(k+N_p|k) \end{bmatrix} \tag{1.20}$$

1.6.2. Calcul de la commande :

La fonction coût à minimiser s'écrit sous la forme :

$$V(k) = \sum_{i=N_1}^{N_p} \|\hat{z}(k+i|k) - r(k+i)\|^2_{Q(i)} + \sum_{i=0}^{N_u-1} \|\Delta\hat{u}(k+i|k)\|^2_{R(i)} \quad (1.21)$$

Avec

$r(k)$: la référence

$Q(i)$et $R(i)$: matrices de pondérations

Cette fonction peut être récrite sous la forme :

$$V(k) = \|Z(k) - T(k)\|^2_{\mathcal{Q}} + \|\Delta U(k)\|^2_{\mathcal{R}} \quad (1.22)$$

Où

$$Z(k) = \begin{bmatrix} \hat{z}(k+N_1|k) \\ \vdots \\ \hat{z}(k+N_p|k) \end{bmatrix}, T(k) = \begin{bmatrix} \hat{r}(k+N_1|k) \\ \vdots \\ \hat{r}(k+N_p|k) \end{bmatrix}, \Delta U(k) = \begin{bmatrix} \Delta\hat{u}(k|k) \\ \vdots \\ \Delta\hat{u}(k+N_u-1|) \end{bmatrix} \quad (1.23)$$

Et les matrices de pondérations $\boldsymbol{\mathcal{Q}}$ et $\boldsymbol{\mathcal{R}}$ sont données par :

$$\mathcal{Q} = \begin{bmatrix} Q(N_1) & 0 & \cdots & 0 \\ 0 & Q(N_1+1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & Q(N_p) \end{bmatrix}; \mathcal{R} = \begin{bmatrix} R(0) & 0 & \cdots & 0 \\ 0 & R(1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R(N_u-1) \end{bmatrix} \quad (1.24)$$

A partir des prédictions, Z peut être écrit sous la forme :

$$Z(k) = \Psi x(k) + Yu(k-1) + \Theta\Delta U(k) \quad (1.25)$$

$$\text{On définit : } \varepsilon(k) = T(k) - \Psi x(k) - \mathrm{Y}u(k-1) \quad (1.26)$$

On obtient :

$$V(k) = const - \Delta U(k)^T G + \Delta U(k)^T H \Delta U(k) \quad (1.27)$$

$$\text{Où: } G = 2\Theta^{\mathrm{T}}\mathcal{Q}\boldsymbol{\theta\varepsilon}(\mathrm{k}) \text{et } H = \Theta^{\mathrm{T}}\,\mathcal{Q}\,\Theta + \mathcal{R} \quad (1.28)$$

Et ni *G* ni *H* dépond de $\Delta U(k)$. Pour calculer $\Delta U(k)$ optimal, il faut calculer le gradient de *V(k).* La solution est alors donnée par :

$$\Delta U(k)_{opt} = \frac{1}{2} H^{-1} G \quad (1.29)$$

Nous utilisons seulement la partie de cette solution qui correspond au premier élément du vecteur, nous pouvons représenter ceci comme :

$$\Delta u(k)_{opt} = [I_l, 0_l, \dots, 0_l]\Delta U(k)_{opt} \quad (1.30)$$

Où: I_l est la matrice d'identité, 0_l est la matrice zéro.

1.7. La commande prédictive linéaire avec contraintes:

On considère le problème de régulation à l'origine du système linéaire discret suivant :

$$x(t+1) = Ax(t) + Bu(t) \quad (1.31)$$

$$y(t) = C_y x(t) \quad (1.32)$$

$$z(t) = C_z x(t) \quad (1.33)$$

Avec :

- $x(t) \in R^n$ est le vecteur d'état, $u(t) \in R^m$ est le vecteur d'entré
- $y(t) \in \mathbb{R}^{m_y}$ est le vecteur des sorties mesurées
- $z(t) \in \mathbb{R}^{m_z}$ est le vecteur des sorties à contrôler
- $A \in R^{n*n}, B \in R^{n*m}$ et l'ensemble (A, B) est contrôlable.

Supposons que, à chaque instant *t,* tout le vecteur d'état x est mesurable, la MPC résout le problème d'optimisation donné par :

$$\min_u \{ J(U, x(t)) = x_{t+N/t}^T P x_{t+N/t} + \sum_{k=0}^{N-1} x_{t+k/t}^T Q x_{t+k/t} + u_{t+k}^T R u_{t+k} \} \quad (1.34)$$

Avec

$$y_{min} \leq y_{t+k|t} \leq y_{max}, k = 1, \dots, N$$

$$u_{min} \leq u_{t+k} \leq u_{max,} k = 0, \dots, M-1$$

$$x_{t|t} = x(t)$$

$$y_{t+k|t} = C x_{t+k|t}, k \geq 0$$

Le vecteur $U = [\hat{u}(t|t)^T, \dots, \hat{u}(t+M-1|t)^T]^T$

Les contraintes doivent être satisfaites sur tout l'horizon de prédiction et de commande, elles peuvent être misent sous la forme :

$$E \begin{bmatrix} \Delta U(t) \\ 1 \end{bmatrix} \leq 0 \quad (1.35)$$

$$F \begin{bmatrix} U(t) \\ 1 \end{bmatrix} \leq 0 \quad (1.36)$$

$$G \begin{bmatrix} Z(t) \\ 1 \end{bmatrix} \leq 0 \quad (1.37)$$

E, F et G sont des matrices de dimensions appropriées, nous pouvons employer des contraintes de cette forme pour représenter le taux de variation du signal de commande entre deux périodes d'échantillonnage, l'intervalle de fonctionnement des actionneurs, et les contraintes possibles sur les sorties à commandées.

Le problème d'optimisation avec contraintes à résoudre à chaque période d'échantillonnage est :

$$\Delta U(k)^T H \Delta U(k) - G^T \Delta U(k) \tag{1.38}$$

Ce problème a la forme générale

$$\begin{cases} \min_\theta \ \frac{1}{2}\theta^T \Phi \theta + \phi^T \theta \\ \Omega\theta \leq \omega \end{cases} \tag{1.39}$$

Qui est un problème standard appelé programme quadratique (*QP)*.

1.7.1. Méthodes de solutions de la commande prédictive linéaire avec contraintes:

Dans cette partie, on va présenter les méthodes les plus classiques et les plus efficaces pour la solution de la commande prédictive linéaire avec contraintes. Ce problème est formulé par l'équation (1.39).

1.7.1.1. Méthode de l'ensemble actif :[11]

L'idée de base des méthodes d'ensemble actif est de définir, à chaque itération de l'algorithme, l'ensemble de contraintes actives et inactives, appelée l'ensemble actif (ou ensemble de travail). Cet ensemble est choisi pour être un sous-ensemble des contraintes qui sont réellement actives au point courant, et donc ce point est faisable pour l'ensemble de travail. Une solution optimale est obtenue par la résolution d'une suite de problèmes de programme quadratique (QP) avec contraintes d'égalité qui peut se résumer par projection à la résolution d'un problème sans contraintes, ce qui constitue un avantage certain en termes de temps de calcul.

1.7.1.2. Les méthodes du point intérieur [12] : Les méthodes du point intérieur deviennent de plus en plus populaires, au détriment des méthodes d'ensemble actif, car leur facteur de convergence est polynomial. Ces algorithmes utilisent une fonction de pénalité intérieure

(barrière) et des algorithmes de type Newton. Leurs performances font qu'ils sont particulièrement adaptés aux problèmes d'optimisation convexe de grandes dimensions.

1.7.1.3. Programmation Quadratique multiparamétrique (MPQP)

Cette méthode conduit à implémenter la commande prédictive par l'intermédiaire de l'évaluation d'une fonction explicite du vecteur d'état en utilisant la programmation quadratique multiparamétrique [5]. Cette fonction est construite hors ligne. Par conséquent, la commande prédictive ne nécessite pas la résolution d'un nouveau problème d'optimisation quand les paramètres changent, car la solution optimale peut être mise à jour en utilisant la fonction pré-calculée. De cette manière, le calcul en ligne est réduit à une évaluation d'une série de fonctions au lieu de l'optimisation en temps réel.

1.7.1.4. Commande prédictive approximée (APC)

Récemment, de nouveaux algorithmes sont introduits basés sur l'approximation de la commande prédictive. La commande prédictive approximée partage le même principe que la commande prédictive explicite [6, 13, 14]. Dans un premier lieu, une loi de commande explicite est calculée hors ligne, ensuite une approximation de cette loi est développée et implantée en ligne.

1.8. La commande prédictive non-linéaire :

La commande prédictive non-linéaire (NMPC) est basée sur des modèles non-linéaires. En raison des défis de garantir une solution globale ou au moins une solution admissible à tout instant garantissant les performances, le nombre d'applications de la commande prédictive non-linéaire est encore limité. Cependant, son potentiel est vraiment grand puisque la NMPC doit faire des perfectionnements dans des secteurs où la non-linéarité des processus est forte et les demandes du marché exigent des changements fréquents en régimes de fonctionnement.

1.8.1. Formulation générale de la NMPC :

On considère un système non-linéaire décrit par le modèle d'état discret suivant :

$$x(k+1) = f\big(x(k), u(k)\big) \tag{1.40}$$

Où x est l'état du système, u son entrée de commande et f est une application continue. Le signal de commande u est sous la contrainte suivante :

$$u(k) \in \mathbb{U} \subset \mathbb{R}^m \tag{1.41}$$

U est un ensemble compact convexe avec $0 \in int(\mathbb{U})$ et $f(0,0) = 0$, aussi l'état peut être mise sous contrainte pour rester à l'intérieur d'un ensemble convexe et fermé :

$$x(k) \in \mathbb{X} \tag{1.42}$$

Le problème résolu par la commande prédictive non-linéaire est un problème de régulation vers l'origine par la résolution du problème d'optimisation suivant :

$$\min_{\mathcal{U}} J_N(x, k, \mathcal{U}) \tag{1.43}$$

Avec (1.40), (1.41) et (1.42)

dont

$$J_N(x, k, \mathcal{U}) = F\big(x(k+N)\big) + \sum_{i=k}^{k+N-1} L\big(x(i), u(i)\big) \tag{1.44}$$

N est l'horizon de prédiction, *F(x(k+N))* : coût sur l'état final $x(k+N)$. Une contrainte finale sur l'état peut aussi être utilisée, elle peut s'écrire d'une façon générale comme suit :

$$x(k+N) \in X_f \subset \mathbb{X} \tag{1.45}$$

Où X_f est un sous-ensemble fermé et convexe de $\mathbb{X}$.

Le coût sur l'état finale *F* et la région finale sont introduits pour garantir la stabilité de la commande prédictive non-linéaire. La solution donne une séquence de commande selon *N* :

$\mathcal{U} = [u(k) \quad u(k+1) \quad \cdots \quad u(k+N-1)] \in \mathbb{U}^N$ et seul la commande *u(k)* est appliquée au système à l'instant *k*. Cette procédure est répétée à chaque période d'échantillonnage.

1.8.2. Solution du problème de la commande prédictive non-linéaire

Le problème d'optimisation de la commande prédictive non-linéaire est généralement non convexe. Ce problème est très souvent résolu en utilisant le programme quadratique séquentiel (SQP), qu'est une extension de la méthode de l'ensemble actif utilisée pour résoudre le programme quadratique [4]. Cette méthode est difficile à programmer et prend beaucoup de temps de calcul. D'autres algorithmes ont été proposés pour la résolution de ce problème dans un temps de calcul raisonnable tel que la méthode de tire multiple (multiple shooting method) [15], et la méthode de la somme non-linéaire des carrés (nonlinear sum of squares)[16][17].

1.9. Stabilité de la commande prédictive

Durant les années soixante-dix et le début des années quatre-vingt, le problème de la stabilité de la commande prédictive n'avait été considéré sérieusement. A partir de cette date,

apparaissent des travaux exploitant des horizons finis et infinis pour lesquels il est possible de démontrer, sous certaines conditions, la stabilité du système contrôlé par une stratégie MPC. Par exemple, dans [18], la fonction coût a été utilisée comme une fonction de Lyapunov pour établir la stabilité des systèmes où une contrainte d'égalité terminale est exigée. Ce résultat a été étendu dans le domaine continu [19]. Dans un contexte non-linéaire, Keerthi et Gilbert [20] sont les premiers qui ont utilisé ce principe pour établir la stabilité de la commande prédictive à base de modèles sur les systèmes non-linéaires en temps discret avec contraintes. Depuis, une floraison de formulations a vu le jour. Aujourd'hui, des conditions de stabilité des schémas de la MBPC sont ainsi disponibles [21] [22].

1.10. Conclusion :

Dans la première section de ce chapitre, nous avons présenté le principe de base de la commande prédictive. Après, les modèles de prédictions linéaires et non-linéaires ont été montrés dans la deuxième partie. Dans la troisième section, la formalisation de la commande prédictive linéaire et non-linéaire a été abordée, et les méthodes de solutions de la commande prédictive linéaire et non-linéaire ont été présentées. Finalement, un aperçu sur la stabilité de la commande prédictive a été donné.

Chapitre 2

Commande Prédictive Robuste par les Intervalles Flous

2.1 Introduction

La théorie des intervalles flous introduite en particulier dans [23] a trouvé des applications intéressantes pour modéliser et commander les systèmes présentant des incertitudes dans les paramètres. Contrairement à un intervalle conventionnel qui inclut seulement toutes les valeurs possibles, un intervalle flou contient, dans sa définition, un degré d'incertitude pour chacune des valeurs possibles.

Dans ce chapitre, les intervalles flous sont utilisés pour décrire les incertitudes structurées. En effet, quelques paramètres du modèle du système sont représentés par des intervalles flous. En se basant sur les alpha-coupes et la sortie contrôlée, un nouveau modèle de prédiction est calculé et employé par la MPC pour définir le signal de commande, à chaque période d'échantillonnage. Cette approche est testée sur deux exemples : la commande de la vitesse angulaire d'un moteur à courant continu et la commande de la concentration dans un réacteur chimique continu agité (CSTR).

2.2. Concepts pertinents et notations [24]

2.2.1. Définition1: Intervalle conventionnel

On définit un nombre de l'intervalle *a* comme l'ensemble des nombres réels *x* tel que :

$$a = [a^-, a^+] = \{x / a^- \leq x \leq a^+, x \in R\} \tag{2.1}$$

Pour deux intervalles $a_1 = [a_1^-, a_1^+]$ et $a_2 = [a_2^-, a_2^+]$, les opérations arithmétiques sur l'intervalle (addition, soustraction, multiplication et division) peuvent être définies comme suit:

Addition :

$$a_1 + a_2 = [a_1^- + a_2^-, a_1^+ + a_2^+] \tag{2.2}$$

Soustraction :

$$a_1 - a_2 = [a_1^- - a_2^+, a_1^+ - a_2^-] \tag{2.3}$$

Multiplication :

$$a_1 * a_2 = [Min(E_1), Max(E_1)] \text{ avec } E_1 = \{a_1^- * a_2^-, a_1^- * a_2^+, a_1^+ * a_2^-, a_1^+ * a_2^+\} \tag{2.4}$$

Division :

$$\frac{a_1}{a_2} = [a_1^-, a_1^+] * \left[\frac{1}{a_2^+}, \frac{1}{a_2^-}\right], \ si\ 0 \notin a_2 \tag{2.5}$$

On définit le point milieu *Mid* (midpoint), le rayon *Rad* (radius) (ou demi largeur) et la fonction d'ampleur relative (zone relative) *Rex* d'un intervalle *a* comme suit:

$$Mid(a) = \frac{1}{2}(a^+ + a^-) \tag{2.6}$$

$$Rad(a) = \frac{1}{2}(a^+ - a^-) \tag{2.7}$$

$$Rex(a) = \frac{Rad(a)}{Mid(a)} \tag{2.8}$$

Un intervalle *a* peut être exprimé comme :

$$a = Mid(a) + Rad(a)[-1, 1] \tag{2.9}$$

2.2.2. Propriétés algébriques principales

L'addition et la multiplication des intervalles sont commutatives et associatives. Soit les intervalles $a_1 = [a_1^-, a_1^+]$, $a_2 = [a_2^-, a_2^+]$ et $a_3 = [a_3^-, a_3^+]$,

$$a_1 + a_2 = a_2 + a_1 \tag{2.10}$$

$$a_1 * a_2 = a_2 * a_1 \tag{2.11}$$

$$a_1 + (a_2 + a_3) = (a_2 + a_1) + a_3 \tag{2.12}$$

$$a_1 * (a_2 * a_3) = (a_2 * a_1) * a_3 \tag{2.13}$$

Les intervalles 0 et 1 sont les éléments d'identité additifs et multiplicatifs, respectivement, dans le système d'intervalles:

$$0 + a = a + 0 = a \tag{2.14}$$

$$1 * a = a * 1 = a \tag{2.15}$$

$$0 * a = a * 0 = 0 \tag{2.16}$$

2.2.3. Non-existence des éléments inverses

Il n'existe pas d'intervalles inverses additifs ou multiplicatifs exceptés des intervalles dégénérés.

L'intervalle *–a* ne représente pas l'inverse additif de l'intervalle *a*

$$a + (-a) = [a^-, a^+] + [-a^+, -a^-] = [a^- - a^+, a^+ - a^-] \tag{2.17}$$

Et similairement pour la division.

Cependant, nous avons toujours les inclusions : $0 \in a-a$ et $1 \in a/a$.

2.2.4. Définition 2 : Les ensembles flous

La notion d'ensemble flou a été proposée par Zadeh (Zadeh, 1965) en introduisant un caractère graduel de l'appartenance d'un élément à un ensemble donné. Cela permet une meilleure représentation des termes et des connaissances vagues que nous, les humains, manipulons au quotidien.

Un sous-ensemble flou A dans un univers du discours X est caractérisé par sa fonction d'appartenance $\mu_A(x)$ qui associe à chaque élément x de X une valeur dans l'intervalle des nombres réels [0, 1] :

$$\mu_A : X \rightarrow [0\ , 1] \tag{2.18}$$

Ainsi un sous-ensemble flou A dans X peut être représenté par un ensemble de couples ordonnés

$$A = \{(x, \mu_A(x)) / x \in X\} \tag{2.19}$$

2.2.5. Définition 3 : L'alfa coupe

Une α-coupe d'un ensemble flou A peut être définie comme suit:

$$A_\alpha = \{x / \mu_A(x) \geq \alpha, x \in X, \alpha \in [0, 1]\} \tag{2.20}$$

Les alpha-coupes d'un ensemble flou définissent des sous-ensembles contenant tous les éléments de l'univers de discours X, ayant un degré d'appartenance à l'ensemble A, supérieur ou égal à la valeur spécifiée de $\alpha \in [0, 1]$.

2.2.6. Définition 4 *:* Les intervalles flous

Un intervalle a est défini comme l'ensemble de nombres réels compris entre une limite inférieure a^- et une limite supérieure a^+ tels que :

$$a = \{x \mid a^- \leq x \leq a^+, x \in R\} \tag{2.21}$$

Ceci peut être représenté par une fonction d'appartenance rectangulaire $\mu_a(x)$. La fonction d'appartenance $\mu_a : R \rightarrow \{0,1\}$ de l'intervalle a est défini par :

$$\mu_a : \begin{cases} 1 & si\ x \in a \\ 0 & autrement \end{cases} \tag{2.22}$$

Cette représentation montre que toutes les valeurs possibles de l'intervalle lui appartiennent avec le même degré d'appartenance.

Dans le cas général d'un intervalle flou *F,* utilisant une représentation trapézoïdale floue. Une fonction d'appartenance est associée à l'intervalle flou F où son support est l'intervalle de toutes les valeurs possibles et dont le noyau est composé de meilleures valeurs de paramètre comme dans la figue 2.1 [25, 26].

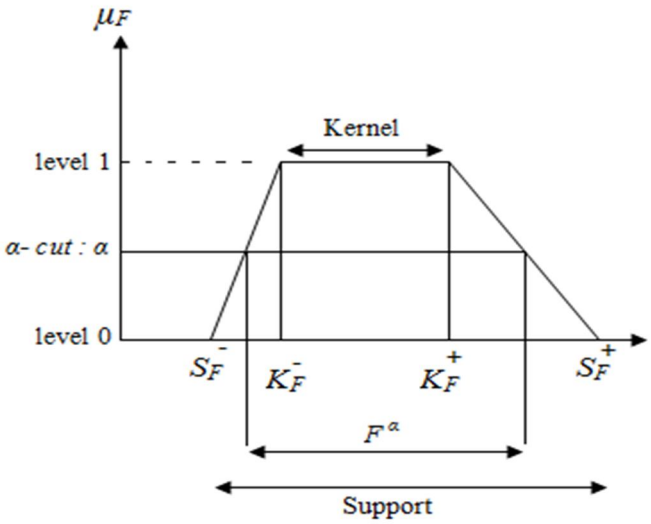


Figure.2.1 : Représentation d'un intervalle flou « trapézoïdal »

La fonction d'appartenance μ_F de l'intervalle *F* est donnée par

$$\mu_F : \begin{cases} R \rightarrow [0,1] \\ x \rightarrow \mu_F(x) \end{cases} \tag{2.23}$$

L'intervalle flou peut être exprimé par son support et son noyau :

$support(F) = [S_F^-, S_F^+]$, et $noyau(F) = [K_F^-, K_F^+]$ pour un intervalle trapézoïdal

Et $support(F) = [S_F^-, S_F^+]$, et $noyau(F) = [K_F]$ pour un intervalle triangulaire

Une α-coupe d'un intervalle flou peut être définie par :

$$F^\alpha = \{x | \mu_F(x) \geq \alpha;\ \alpha \in]0,1]\} \tag{2.24}$$

Les intervalles flous peuvent être considérés comme une généralisation de l'intervalle conventionnel par l'addition d'une autre dimension verticale correspondant au degré d'appartenance de chaque élément.

Une α-coupe d'un intervalle flou *F* peut être définie par l'intervalle composé par les éléments qui ont un degré d'appartenance supérieur ou égal à la valeur du α considérée:

$$F^\alpha = [F^{\alpha-}, F^{\alpha+}] \tag{2.25}$$

2.3. Modèle d'état par les intervalles flous

Le modèle d'état basé sur les α-coupes des intervalles flous peut être introduit comme:

$$x^{\alpha}(k+1) = A^{\alpha}x(k) + B^{\alpha}u(k) \tag{2.26}$$

Avec $A^{\alpha}(n \times n)$, $B^{\alpha}\ (n \times m)$ des matrices avec des éléments a_{ij} défini sur les α-coupes d'un intervalle flou triangulaire F_{ij} avec $noyau(F_{ij}) = a_{ijnom}$, support $(F_{ij}) = [\, a_{ijmin} a_{ijmax}]$, l'α-coupe de l'intervalle flou $F_{ij}^{\alpha} = [\, a_{ij}^{\alpha-}\ a_{ij}^{\alpha+}\,]$, et des éléments b_{ij} défini sur les α-coupes d'un intervalle flou triangulaire F'_{ij} avec $noyau(F'_{ij}) = b_{ijnom}$, support $(F'_{ij}) = [\, b_{ijmin} b_{ijmax}]$, et le α-coupe de l'intervalle flou $F'^{\alpha}_{ij} = [\, b_{ij}^{\alpha-}\ b_{ij}^{\alpha+}\,]$, $x(k) \in R^n$ et $u(k) \in R^m$ sont donnés à l'instant k et les composants de l'état $x^{\alpha}(k+1)$ sont défini sur l'α-coupe de l'intervalle flou X_i:

$$X_i^{\alpha}(k+1) = [x_i^{\alpha-}(k+1)\ \ x_i^{\alpha+}(k+1)] \tag{2.27}$$

Les composantes étant obtenues comme :

$$x_i^{\alpha-}(k+1) = \sum_j a_{ij}^{\alpha-} x_j(k) + \sum_l b_{il}^{\alpha-} u_l(k) \tag{2.28}$$

Et

$$x_i^{\alpha+}(k+1) = \sum_j a_{ij}^{\alpha+} x_j(k) + \sum_l b_{il}^{\alpha+} u_l(k) \tag{2.29}$$

Les paramètres : a_{ijnom}, $a_{ijmin}, a_{ijmax}, b_{ijnom}, b_{ijmin}\ et\ b_{ijmax}$ sont défini par un algorithme d'identification.

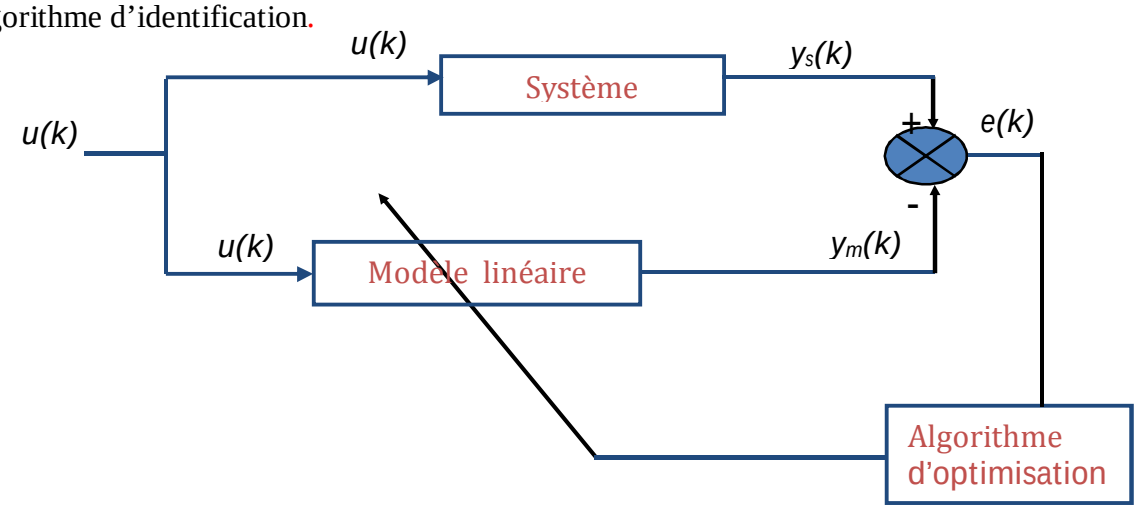


Figure.2.2 : Schéma de base de l'algorithme d'identification

L'identification consiste à choisir une forme de modèle pour le système, définir un critère qui permet de faire correspondre y_m et y_s et de choisir une technique d'optimisation.
Par l'utilisation du schéma précédent : trois modèles sont obtenus :

- modèle nominal du système (a_{ijnom}, b_{ijnom})
- modèle du système avec une variation de +Δ% sur les paramètres (a_{ijmax}, b_{ijmax})
- modèle du système avec une variation de -Δ% sur les paramètres (a_{ijmin}, b_{ijmin})

2.3.1. Propriété 1:

Les intervalles flous X_i sont des ensembles flous triangulaires avec

$noyau(X_i) = x_{inom}(k+1), support(X_i) = [x_{imin}(k+1) x_{imax}(k+1)]$

avec :

$$x_{inom}(k+1) = \sum_j a_{ijnom} x_j(k) + \sum_l b_{ilnom} u_l(k) \tag{2.30}$$

Et $x_{imin}(k+1)$, $x_{imax}(k+1)$ obtenus de la même manière.

2.3.2. Démonstration:

D'après la définition ci-dessus de l'α-coupe et la fonction d'appartenance triangulaire nous avons

$$\alpha = \mu\left(a_{ij}^{\alpha-}\right) = \left(\frac{a_{ij}^{\alpha-} - a_{ijmin}}{a_{ijnom} - a_{ijmin}}\right) \tag{2.31}$$

ET

$$\alpha = \mu\left(a_{ij}^{\alpha+}\right) = \left(\frac{a_{ij}^{\alpha+} - a_{ijmax}}{a_{ijnom} - a_{ijmax}}\right) \tag{2.32}$$

$$\alpha = \mu\left(b_{ij}^{\alpha-}\right) = \left(\frac{b_{ij}^{\alpha-} - b_{ijmin}}{b_{ijnom} - b_{ijmin}}\right) \tag{2.33}$$

Et

$$\alpha = \mu\left(b_{ij}^{\alpha+}\right) = \left(\frac{b_{ij}^{\alpha+} - b_{ijmax}}{b_{ijnom} - b_{ijmax}}\right) \tag{2.34}$$

Par le réarrangement de (2.31) et (2.32) on obtient

$$a_{ij}^{\alpha-} = \left(a_{ijnom} - a_{ijmin}\right)\alpha + a_{ijmin} \tag{2.35}$$

et

$$b_{il}^{\alpha-} = \left(b_{ilnom} - b_{ilmin}\right)\alpha + b_{ilmin} \tag{2.36}$$

Par le remplacement dans (2.28), on obtient

$$x_i^{\alpha-}(k+1) = \sum_j \left((a_{ijnom} - a_{ijmin})\alpha + a_{ijmin}\right) x_j(k) + \sum_l \left((b_{ilnom} - b_{ilmin})\alpha + b_{ilmin}\right) u_l(k) \quad (2.37)$$

De même, le réarrangement (2.33) et (2.34) et en utilisant (2.29), on obtient:

$$x_i^{\alpha+}(k+1) = \sum_j \left((a_{ijnom} - a_{ijmax})\alpha + a_{ijmax}\right) x_j(k) + \sum_l \left((b_{ilnom} - b_{ilmax})\alpha + b_{ilmax}\right) u_l(k) \quad (2.38)$$

Maintenant:

$$x_i^{\alpha-}(k+1) = \alpha \sum_j a_{ijnom} x_j(k) + \alpha \sum_l b_{ilnom} u_l(k) - \alpha \sum_j a_{ijmin} x_j(k) - \alpha \sum_l b_{ilmin} u_l(k) + \sum_j a_{ijmin} x_j(k) + \sum_l b_{ilmin} u_l(k) \quad (2.39)$$

Qui peut être écrite sous la forme :

$$x_i^{\alpha-}(k+1) = \alpha x_{inom}(k+1) - \alpha x_{imin}(k+1) + x_{imin}(k+1) \quad (2.40)$$

Avec

$$x_{inom}(k+1) = \sum_j a_{ijnom} x_j(k) + \sum_l b_{ilnom} u_l(k) \quad (2.41)$$

$$x_{imin}(k+1) = \sum_j a_{ijmin} x_j(k) + \sum_l b_{ilmin} u_l(k) \quad (2.42)$$

Utilisant le même développement, avec:

$$x_i^{\alpha+}(k+1) = \alpha x_{inom}(k+1) - \alpha x_{imax}(k+1) + x_{imax}(k+1) \quad (2.43)$$

Avec

$$x_{imax}(k+1) = \sum_j a_{ijmax} x_j(k) + \sum_l b_{ilmax} u_l(k) \quad (2.44)$$

Nous observons que $x_i(k+1)$ est une fonction linéaire de α et en réarrangeant:

$$\alpha = \frac{x_i^{\alpha-}(k+1) - x_{imin}(k+1)}{x_{inom}(k+1) - x_{imin}(k+1)} \quad (2.45)$$

Et

$$\alpha = \frac{x_i^{\alpha+}(k+1) - x_{imax}(k+1)}{x_{inom}(k+1) - x_{imax}(k+1)} \quad (2.46)$$

Ainsi les intervalles flous X_i sont des ensembles flous triangulaires avec

$noyau(X_i) = x_{inom}(k+1)$, $Support(X_i) = [x_{imin}(k+1)\ x_{imax}(k+1)]$.

Cette propriété sera utilisée pour construire une MBPC robuste basée sur l'intervalle flou.

2.4. MPC Robuste

L'idée de la commande prédictive robuste est basée sur la propriété 1. Elle consiste à retrouver le modèle du système avec des paramètres incertains définis sur un intervalle flou en calculant α à partir des mesures obtenues. Le modèle du système à commander est donné par l'équation (2.26). Les noyaux et supports des intervalles flous F_{ij} et F'_{ij} sont connus et définis par l'utilisateur. A un instant *k*, deux signaux de commande sont calculés à la fois : la commande nominale ($U_{nom}(k)$) calculée à travers le modèle nominal du système et la commande du système ($U_s(k)$) calculée à travers le modèle de prédiction interne du système qui correspond à l'α-coupe donnée par les équations (2.45) ou (2.46). Le signal de commande $U_s(k)$ est appliqué au système pour obtenir la variable d'état $x(k+1)$, également le signal de commande $U_{nom}(k)$ est appliqué à l'équation (2.41), (2.42) et (2.44) pour obtenir $x_{inom}(k+1)$, $x_{imin}(k+1)$ et $x_{imax}(k+1)$. A partir de ces informations, α est calculé par l'équation (2.45) ou (2.46), le modèle de prédiction du système est pris en tant que (2.40) si

$x(k+1) < x_{inom}(k+1)$ ou (2.43) si $x(k+1) \geq x_{inom}(k+1)$.

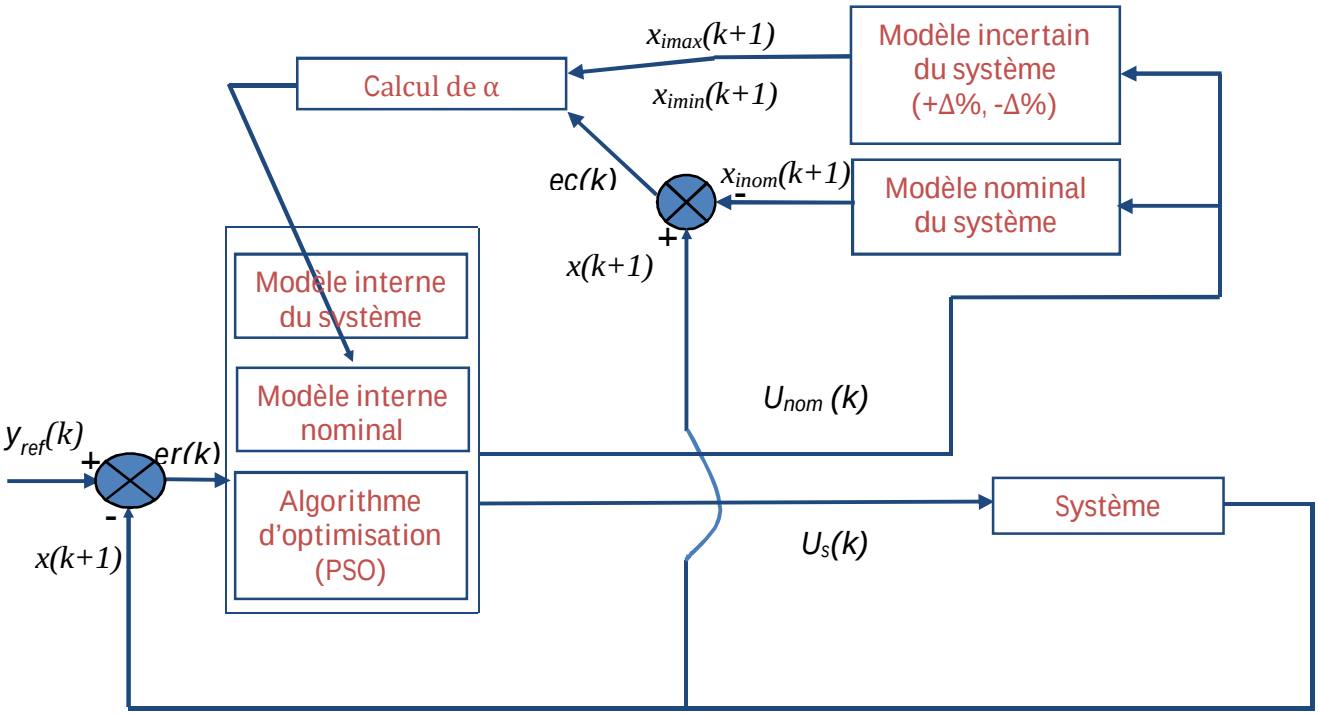


Figure.2.3: Schéma de la commande prédictive robuste

2.5. Résultats de simulation :

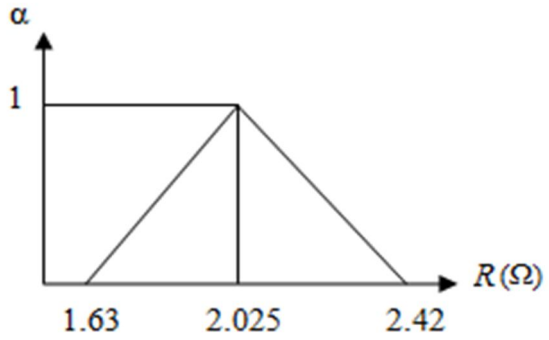
2.5.1. Exemple 1 (système linéaire) :

Dans cette partie, la MBPC robuste basée sur l'intervalle flou est appliquée pour le contrôle de la vitesse angulaire d'un moteur à courant continu avec la présence d'incertitudes sur les paramètres et des perturbations sur la charge. Le modèle du moteur à courant continu est donné par le modèle d'espace d'état représenté par (2.47):

$$\begin{bmatrix} \frac{di}{dt} \\ \frac{d\omega}{dt} \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & -\frac{k_e}{L} \\ \frac{k_m}{j} & -\frac{k_d}{j} \end{bmatrix} \begin{bmatrix} i(t) \\ \omega(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} u(t) - \begin{bmatrix} 0 \\ \tau_r \end{bmatrix} \qquad (2.47)$$

Les paramètres nominaux du moteur sont: *R=2.025 Ω, L=0.5 mH, J=0.2 kg.m*^{2}*;* k_m*=0.2Nm/A ;* k_d*=0.05 N.s/rad,* k_e*=0.2V.s/rad.*

La résistance *R* et la self *L* sont des paramètres incertains. Les incertitudes sont représentées par des intervalles flous avec des fonctions d'appartenance triangulaires comme le montre la figure 2.2.



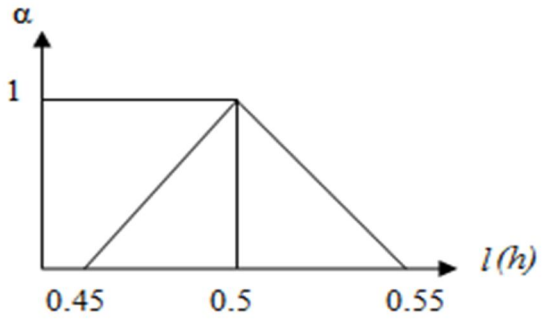


Figure.2.4.a : Intervalle flou de *R* **Figure.2.4.b:**Intervalle flou de *L*

La vitesse angulaire nominale ω_{nom}, minimale ω_{min}et maximale ω_{max}sont obtenus par : (R_{nom}, L_{nom}), (R_{min}, L_{max}), et (R_{max}, L_{min})respectivement.
$\omega(t)$ appartienne à l'intervalle floue Ω où:

$$\text{Kernel}(\Omega) = \omega_{nom}(k+1), \text{ et } \text{Support}(\Omega) = [\omega_{min}(k+1)\omega_{max}(k+1)] \qquad (2.48)$$

Dans les simulations, des variations sont introduites sur *R* et *L* où :

$$R \in [1.63 \;\; 2.42] \quad \text{et} \quad L \in [0.45 \;\; 0.55] \qquad (2.49)$$

Le signal de commande est limité à $50V \leq u \leq 300V$

Les résultats des simulations sont présentés dans les figures 2.5 à 2.9. Dans le cas d'une commande prédictive simple (non robuste), représenté dans la figure 2.6 on remarque que l'effet des variations des paramètres et de la charge de perturbation est très apparent sur la vitesse angulaire qui oscille et ne peut pas suivre la trajectoire de référence.

Le signal de commande calculé pour le contrôle du système incertain est différent de celui calculé pour contrôler le système nominal et est très actif. Dans le cas de la MBPC robuste, illustré dans la figure 2.8, l'effet des oscillations de la vitesse angulaire sont pratiquement éliminé et on peut observer le bon suivi de la trajectoire de référence. Le système est tout à fait commandé et insensible aux variations des paramètres démontrant une bonne propriété de robustesse. De plus, dans ce cas, le signal de commande est proche du signal de commande calculé pour le système nominal. On remarque également que dans tous les cas, les contraintes sur le signal de commande sont respectées.

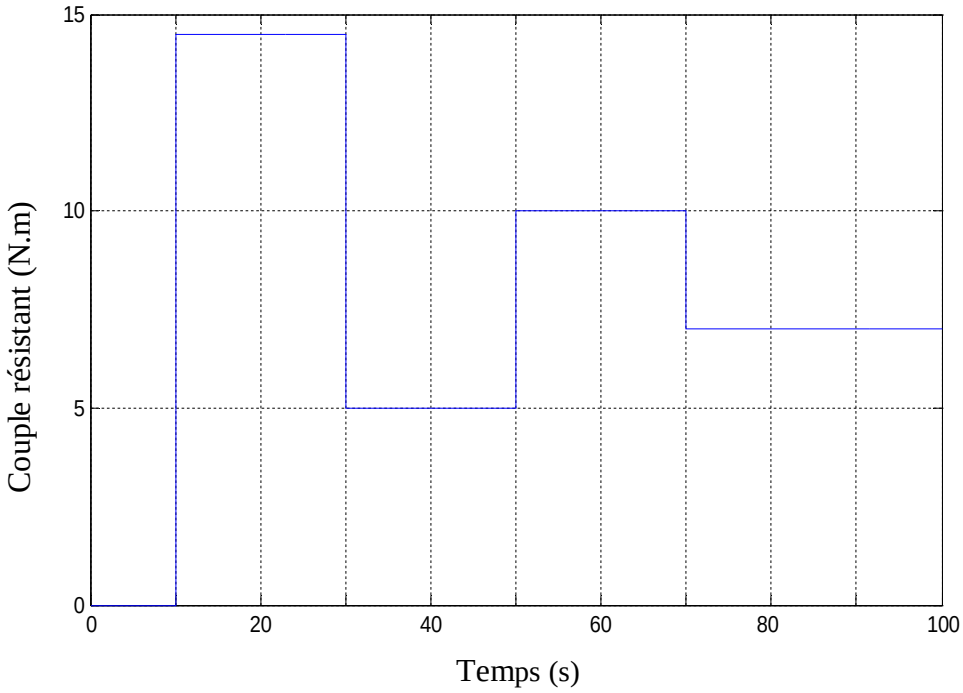


Figure.2.5 : Perturbation sur la charge

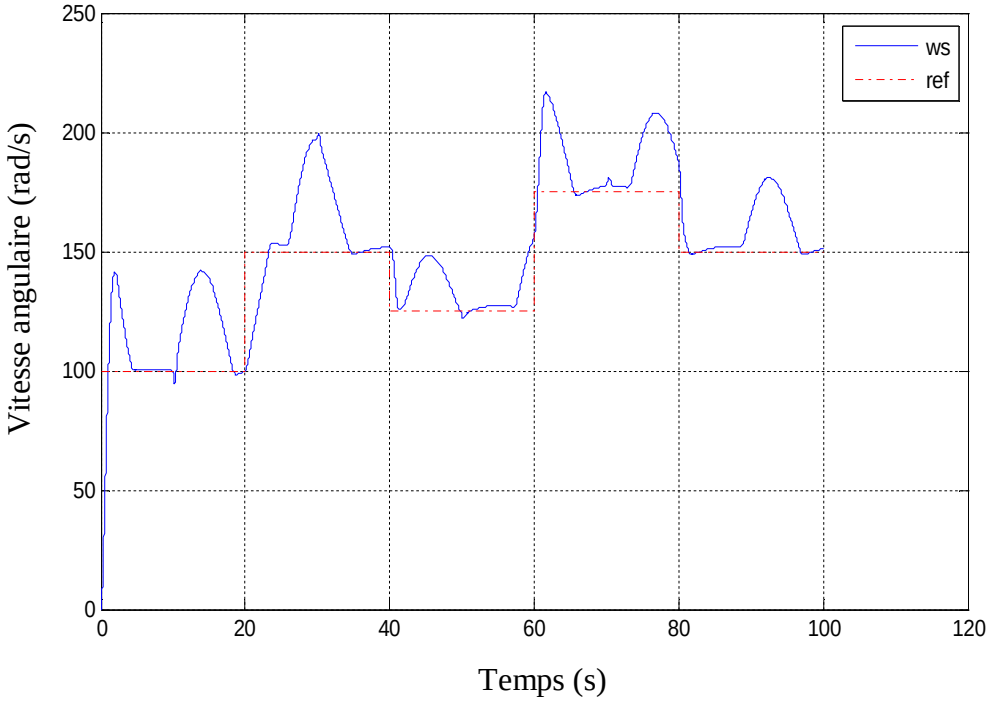


Figure.2.6 : La vitesse angulaire dans le cas de la MBPC simple

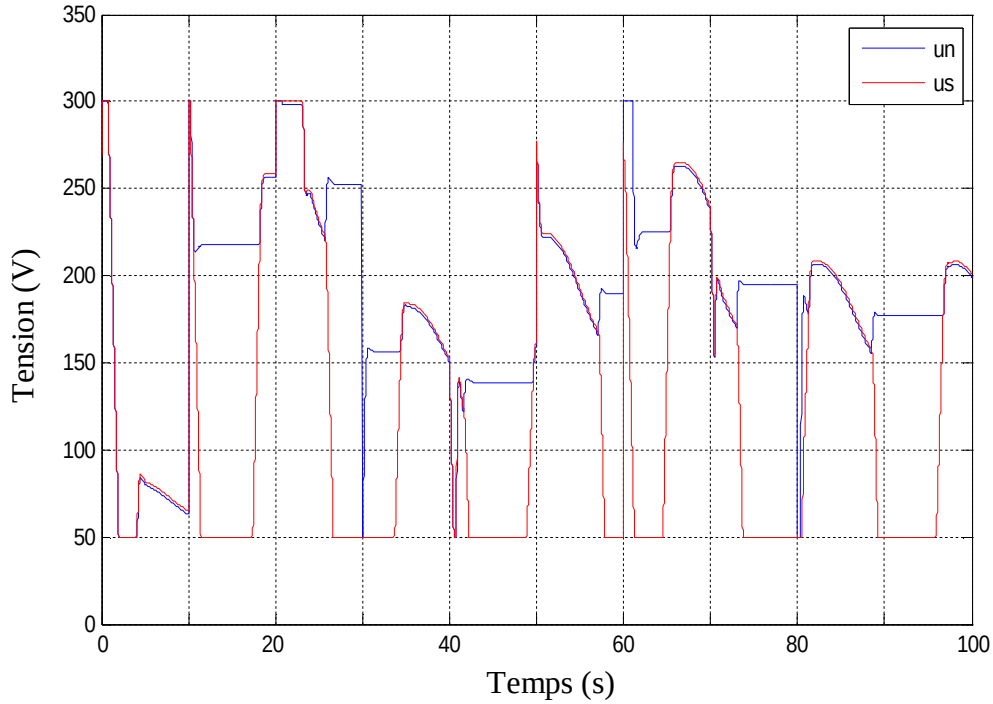


Figure.2.7: Signal de commande dans le cas de la MBPC simple

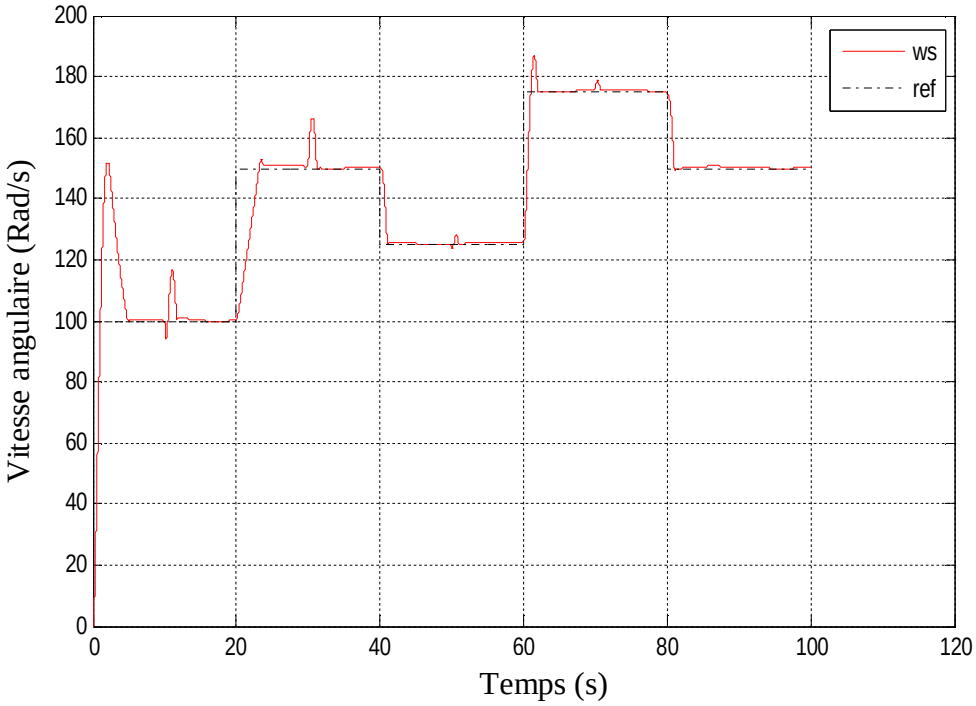


Figure.2.8: Vitesse angulaire dans le cas d'une MBPC robuste avec une variation dans R, L et une perturbation sur la charge

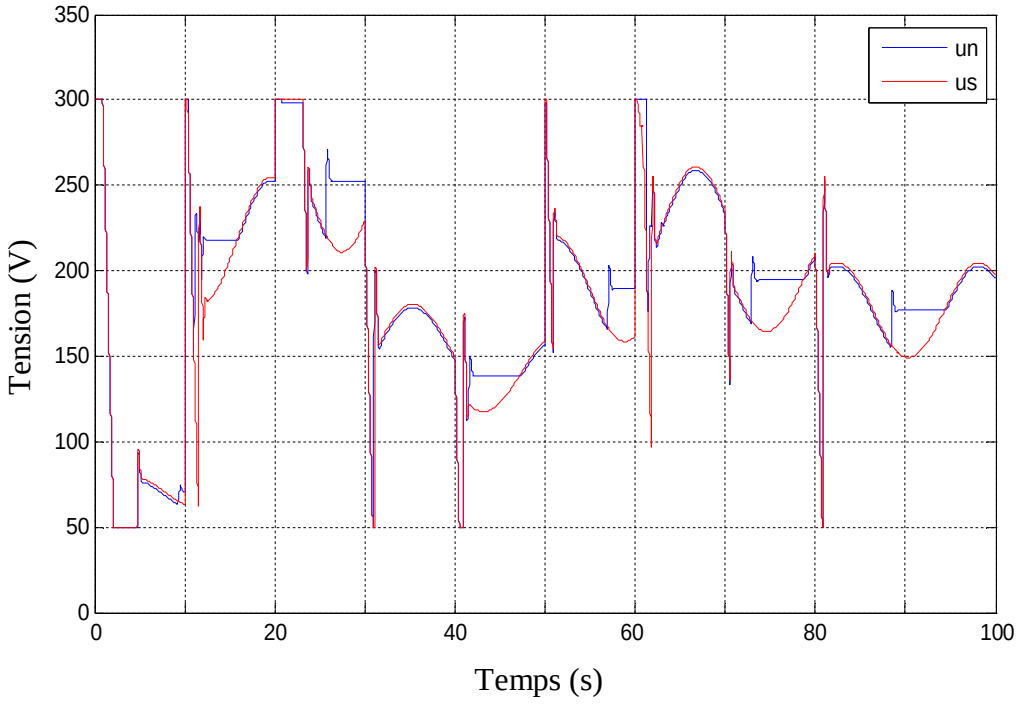


Figure.2.9: Signal de commande dans le cas de la MBPC robuste

2.5.2. Exemple 2 (système non-linéaire) :

On considère l'application de la MBPC robuste basée sur l'intervalle flou à la commande de la concentration dans le CSTR, en présence d'incertitudes dans les paramètres. Le CSTR est un processus non-linéaire qui peut être trouvé couramment dans l'industrie chimique. La concentration et la température dans le CSTR sont décrites par les équations différentielles non-linéaires suivantes [27]:

$$\dot{C_a} = \frac{Q}{V}(C_{a0} - C_a) - a_0 C_a e^{\frac{-E}{RT_a}} \tag{2.50}$$

$$\dot{T_a} = \frac{Q}{V}(T_f - T_a) + a_1 C_a e^{\frac{-E}{RT_a}} + \quad a_3 q_c \left[1 - e^{\frac{-a_2}{q_c}}\right](T_{cf} - T_a) \tag{2.51}$$

a_2 et a_3 sont des paramètres incertains avec des incertitudes représentées par des intervalles flous où les fonctions d'appartenance sont triangulaires (figure 2.10).

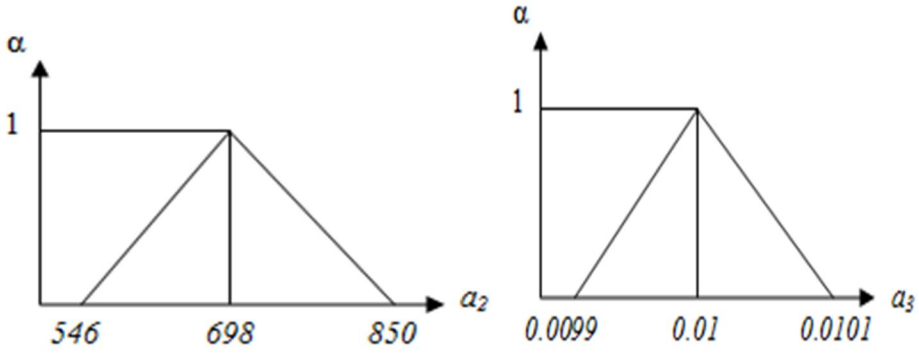


Figure.2.10.a : Intervalle flou de a_2 **Figure.2.10.b :** Intervalle flou de a_3

La concentration nominale, minimale et maximale, $C_{a\,nom}$, $C_{a\,min}$et $C_{a\,max}$, sont obtenues avec(a_{2nom}, a_{3nom}),(a_{2min}, a_{3min}), et (a_{2max}, a_{3max})respectivement. $C_a(k)$ appartient à l'intervalle floue Ω avec:

$$\text{Kernel}(\Omega) = C_{anom}(k+1),\ \text{Support}(\Omega) = [C_{amin}(k+1)\ C_{amax}(k+1)].$$

Des variations sont introduites sur a_2 et a_3 comme suit :

$$a_2 \in [546 \quad 850] \tag{2.52}$$

$$a_3 \in [0.0099 \ \ 0.0101] \tag{2.53}$$

Le signal de commande est limité à $94(l/min) \leq q_c \leq 120(l/min)$.

Le modèle linéaire utilisé par la commande prédictive robuste a la forme :

$$y(k+1) = AI \times y(k) + BI \times u(k) + CI \times y(k-1) + DI \qquad (2.54)$$

y est l'intervalle flou de la concentration C_a. AI, BI, CI, DI sont des intervalles flous, ont les mêmes fonctions d'appartenance que a_2 et a_3. Les Supports et les noyaux de ces paramètres sont obtenus par l'opération de modélisation (offline) du CSTR.

Les résultats de simulations sont présentés sur les figures 2.11 à 2.14. Dans le cas où la commande prédictive simple est appliquée pour la commande du CSTR, l'effet des variations des paramètres est très apparent sur la concentration, qui oscille et ne peut pas suivre la trajectoire de référence (figure 2.11). Dans le cas où le contrôleur MBPC robuste est appliqué, on observe le bon suivi de la trajectoire de référence, comme le montre la figure 2.13. On remarque également que dans tous les cas, les contraintes sur le signal de commande sont respectées. Ces résultats confirment ceux obtenus dans le premier exemple. Le système non-linéaire contrôlé est tout à fait insensible à ces paramètres de grandes variations démontrant une bonne propriété de robustesse.

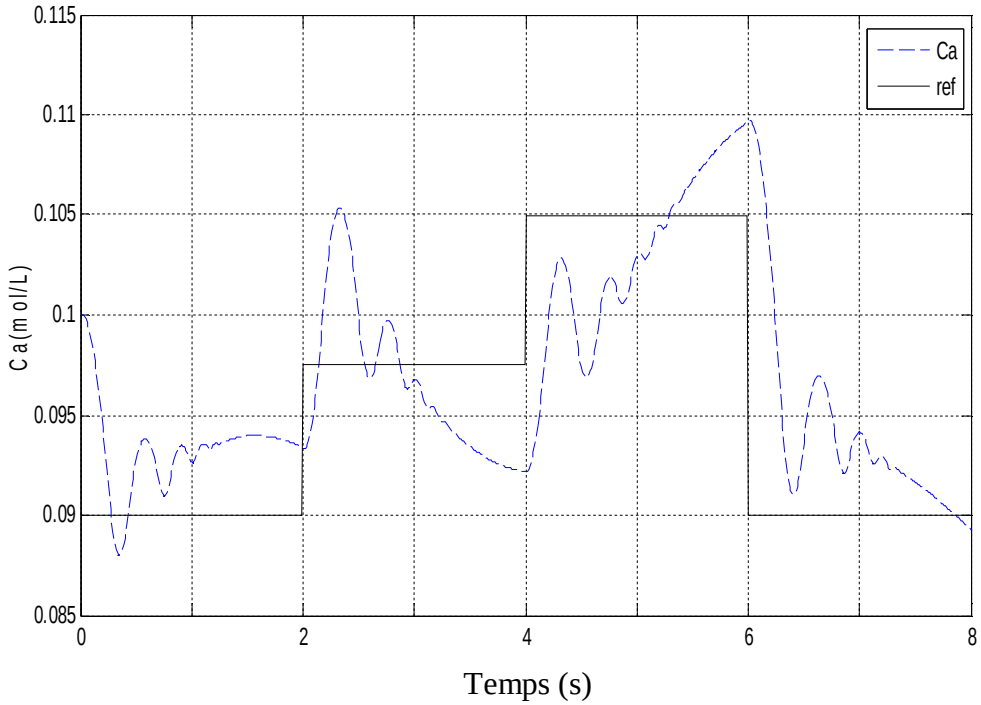


Figure.2.11: Concentration dans le cas de MBPC non robuste avec des variations sur les paramètres

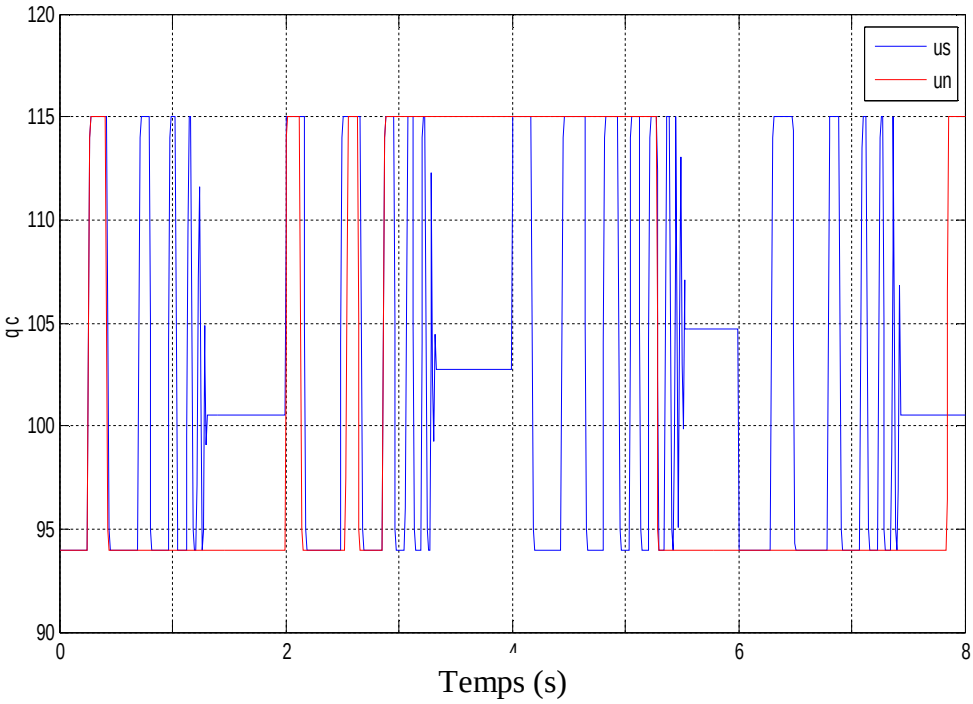


Figure.2.12 : Signal de commande dans le cas de MBPC non robuste

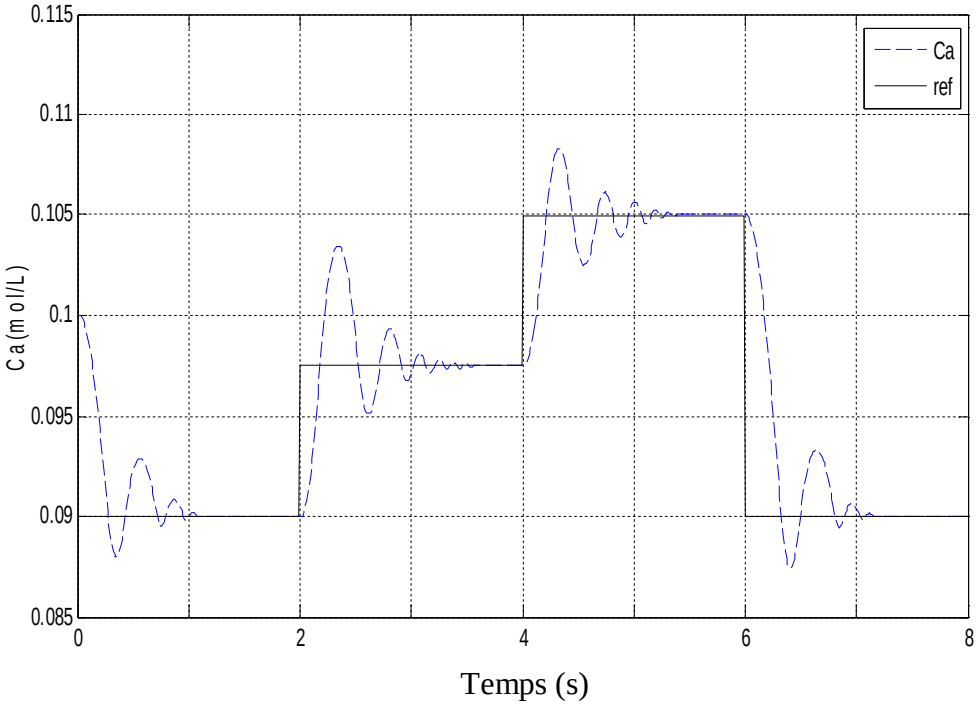


Figure.2.13 : Concentration dans le cas de MBPC robuste

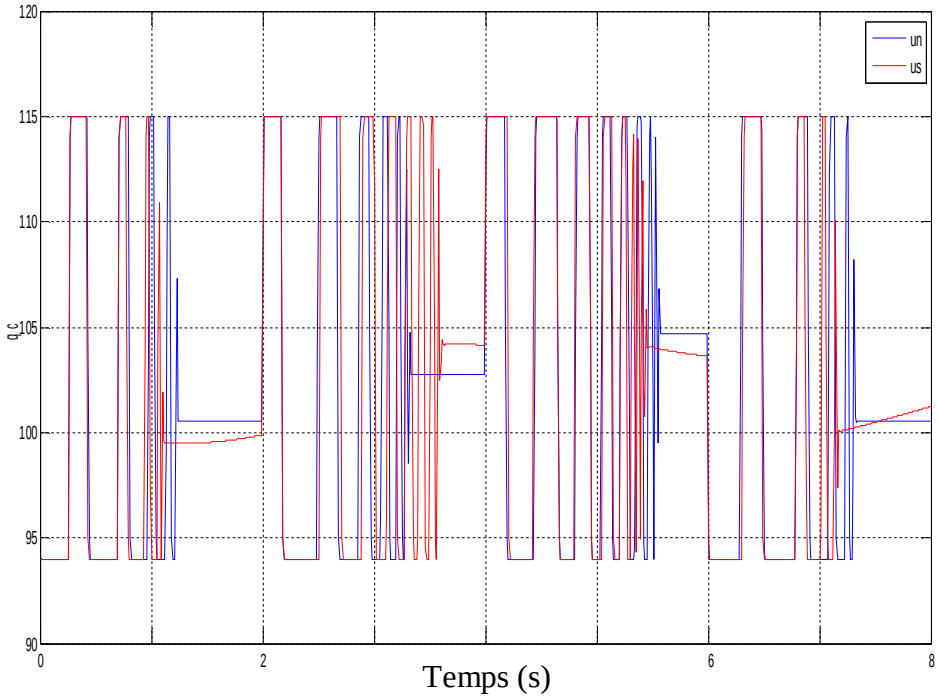


Figure.2.14 : Signal de commande dans le cas de MBPC robuste

2.6. Conclusion :

Dans ce chapitre, nous avons présenté une commande prédictive robuste basée sur un modèle de prédiction avec des incertitudes sur les paramètres décrites par des intervalles flous. Un contrôleur prédictif et la technique des α -coupes ont été utilisés pour obtenir la solution du problème d'optimisation résultant. Cette méthode a été appliquée, dans une étude par simulation, pour commander la vitesse angulaire d'un moteur à courant continu et la concentration dans un réacteur chimique. Afin de montrer l'efficacité de l'approche proposée, cette technique a été comparée avec la commande prédictive non robuste (simple) qui ne tient pas en compte les variations sur les paramètres.

Les résultats obtenus ont montrés l'insensibilité des systèmes commandés aux variations des paramètres démontrant la robustesse de cette approche.

Chapitre 3

Solution de la Commande Prédictive par les Métaheuristiques

3.1. Introduction

Dans la commande prédictive, la qualité de la solution du problème d'optimisation influe directement sur les performances du système attendues. Elles sont meilleures si la solution est proche de l'optimum global. Le choix de la méthode d'optimisation est alors crucial. Ces dernières années une attention particulière a été accordée à ce problème d'optimisation. Le but est de développer des algorithmes puissants permettant de trouver la meilleure solution (optimum global) en un temps de calcul minimal. Récemment, plusieurs chercheurs se sont orientés vers une nouvelle classe de méthodes, nommées métaheuristiques qui sont des algorithmes proposés pour résoudre des problèmes d'optimisation difficiles pouvant donner rapidement une bonne approximation de l'optimum global. Elles se distinguent en cela des méthodes dites exactes, qui garantissent certes la résolution d'un problème, mais avec un temps de calcul parfois prohibitifs.

Parmi cette classe, on trouve des métaheuristiques qui manipulent une population de solutions comme : les algorithmes génétiques (genetic algorithm, GA), l'optimisation par essaim de particules (particle swarm optimization, PSO), l'optimisation par colonies de fourmis (ant colony optimization, ACO), l'algorithme de recherche gravitationnel (gravitational search algorithm, GSA), etc.

Dans ce chapitre, nous considérons l'application et la comparaison des algorithmes PSO, ACO et GSA pour la solution du problème d'optimisation de la commande prédictive. La fonction coût de la commande prédictive (linéaire ou non-linéaire) sera considérée comme une fonction objectif à minimiser par les trois métaheuristiques. La première partie de ce chapitre sera consacrée à la présentation de ces algorithmes, dans la seconde, nous présenterons les applications à la commande prédictive linéaire et non-linéaire avec comparaison en termes de temps de calcul.

3.2. Optimisation par essaim de particules (PSO)

L'optimisation par essaim de particules (de l'anglais Particle Swarm Optimization) est une technique d'optimisation développée par Kennedy et Eberhart en 1995 [8]. Cette méthode est inspirée du comportement social des animaux évoluant en essaim tels que les nuées d'oiseaux, les bancs de poissons et même du comportement social humain [41].

Le PSO est un outil d'optimisation efficace, qui pourrait être implémenté facilement pour résoudre divers problèmes d'optimisation [60, 61].

Un essaim de particules, qui sont les solutions potentielles au problème d'optimisation, survole l'espace de recherche, à la recherche de l'optimum global. Au départ, les particules sont placées aléatoirement dans l'espace de recherche du problème. Dans un espace de recherche de dimension d, à chaque itération, chaque particule se déplace en fonction de trois composantes (figure3.1) :

- vitesse actuelle de la particule (composante d'inertie)
- meilleur site (solution) par lequel elle est déjà passée (composante cognitive)
- meilleur site déjà atteint collectivement par l'essaim (composante sociale)

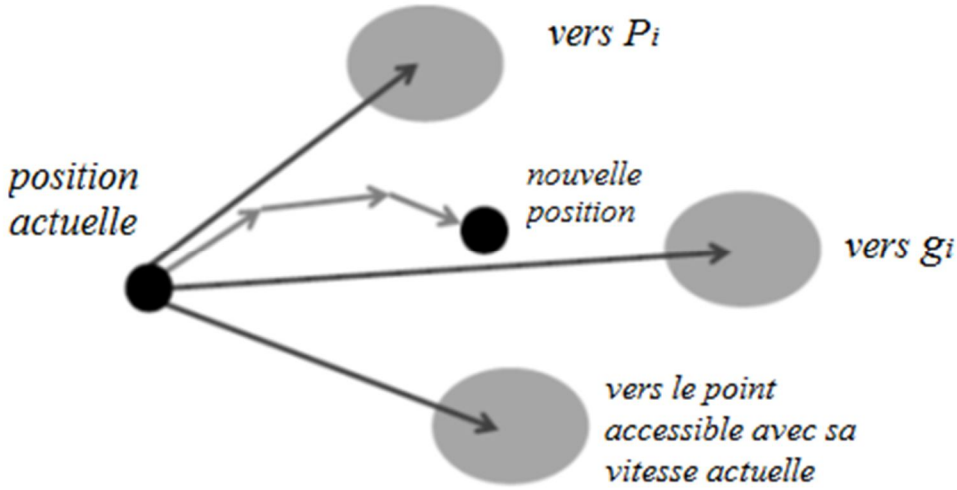


Figure.3.1 : Schéma de principe du déplacement d'une particule

.

Chaque particule *i* de l'essaim est caractérisée par un vecteur de position $\overrightarrow{x_i} = (x_{i1}x_{i2} \dots x_{id})^T$ et un vecteur de vitesse $\overrightarrow{v_i} = (v_{i1}v_{i2} \dots v_{id})^T$. La qualité de sa position est déterminée par la valeur de la fonction coût en ce point. La particule mémorise la meilleure position déjà atteinte $\overrightarrow{p_i} = (p_{i1}p_{i2} \dots p_{id})^T$. La meilleure position, trouvée par toutes les particules de l'essaim, est notée $\overrightarrow{g_i} = (g_{i1}g_{i2} \dots g_{id})^T$.

À chaque instant *t*, le déplacement de chaque particule *i* est manipulé selon les équations suivantes :

$$v_{ij}(t+1) = \omega v_{ij}(t) + c_1 r_1 \left(p_{ij}(t) - x_{ij}(t)\right) + c_2 r_2 \left(g_{ij}(t) - x_{ij}(t)\right), j \in \{1,2, \dots, d\} \quad (3.1)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1), j \in \{1,2, \dots, d\} \quad (3.2)$$

Dont :

v_{ij} : Vitesse de la particule *i* selon la dimension *j*

x_{ij} : Position de la particule *i* selon la dimension *j*

ω : Coefficient d'inertie, généralement constant

c_1, c_2 : Coefficients d'accélérations, constant

r_1, r_2 : Deux nombres aléatoires tirés uniformément dans [0, 1]

$p_{ij}(t)$: Meilleure position par laquel la particule *i* est déjà passée

$g_{ij}(t)$: Meilleur site déjà trouvé par l'essaim

Ces deux équations comprennent les trois composants pour la définition du déplacement de chaque particule de l'essaim où :

$v_{ij}(t)$: est la composante d'inertie, contrôlée par le paramètre ω

$c_1 r_1(p_{ij}(t) - x_{ij}(t))$: est la composante cognitive, contrôlée par le paramètre c_1

$c_2 r_2(g_{ij}(t) - x_{ij}(t))$: est la composante sociale, contrôlée par le paramètre c_2

Le pseudo-code pour l'algorithme d'optimisation par essaim de particules est donné par l'algorithme 1.

Algorithme 1 : optimisation par essaim de particules

01. **Initialisation** de la taille de l'essaim de particules et autre paramètres.
02. **Initialisation** aléatoire des positions et vitesses de toutes les particules.
03. **Pour** chaque particule *i*, $\overrightarrow{p_i} = \overrightarrow{x_i}$
04. **Calculer** le $\overrightarrow{g_i}$
05. **Tant que** (critère d'arrêt n'est pas atteint)**faire**
06. $t = t + 1$;
07. **Calculer** la valeur de la fonction coût de chaque particule ;
08. **Mettre** à jour $\overrightarrow{p_i}$ et $\overrightarrow{g_i}$
09. **Déplacer** les particules selon (3.1) et (3.2)
10. **Fin**

L'algorithme d'optimisation par essaim de particules commence par une initialisation aléatoire de l'essaim. A chaque itération de l'algorithme, les positions des particules sont évaluées, les $\overrightarrow{p_i}$ ainsi que $\overrightarrow{g_i}$ sont mis à jour et les particules sont déplacées selon les formules (3.1) et (3.2). Le critère d'arrêt peut être un nombre maximum d'itérations, une certaine précision (si l'optimum global est connu) ou d'autres critères selon le problème posé.

3.3. Optimisation par colonie de fourmis(ACO)

L'algorithme ACO (de l'anglais Ant Colony Optimization), est une métaheuristique développée par Marco Dorigo en 1992 [9]. Les colonies de fourmis résolvent naturellement des problèmes relativement complexes. Les biologistes ont étudié comment les fourmis arrivent à résoudre collectivement les problèmes de choix lors de l'exploitation de sources de nourriture. Il a été trouvé que les fourmis communiquent entre elles indirectement, par le dépôt sur le sol de substances chimiques volatiles, appelées phéromones, formant des pistes odorantes pour marquer leur trajet entre le nid et une source de nourriture. Ce type de communication indirecte s'appelle stigmergie. Les fourmis qui sont retournées le plus rapidement au nid en passant par la source de nourriture sont celles qui ont emprunté le chemin le plus court et la quantité de phéromone présentée sur ce chemin est légèrement plus importante que celle présentée sur le chemin le plus long ce qui le rend (le chemin le plus court) plus attirant pour les fourmis. Ceci qui est illustré sur la figure 3.2.

L'algorithme de colonie de fourmis a été proposé initialement pour résoudre des problèmes d'optimisations combinatoires (à variables entières) telles que le problème de voyageur de commerce et le problème de sac à dos.

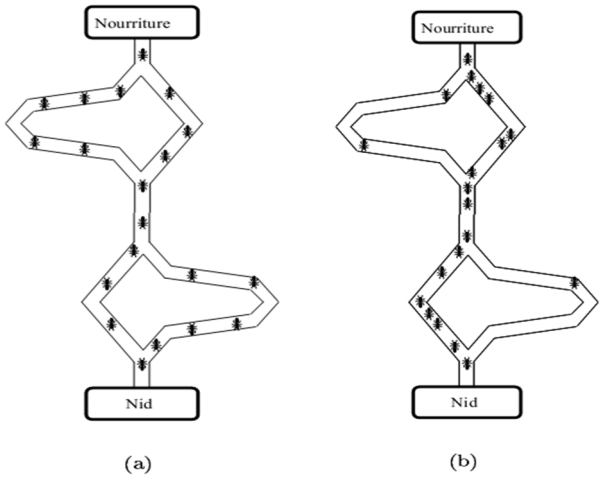


Figure.3.2 : expérience pour la sélection du chemin le plus court :
(a) Au début (b) à la fin

Nous considérons en premier lieu, l'algorithme ACO pour les problèmes à variables entières. Soit $p_{ij}^{k}(t)$ la probabilité qu'à l'itération t la fourmi k choisisse d'aller de la position i à la position j, alors on a :

$$\begin{cases} p_{ij}^{k}(t) = \frac{(\tau_{ij}(t))^{\alpha}.(\eta_{ij})^{\beta}}{\sum_{l \in J_i^k}(\tau_{il}(t))^{\alpha}.(\eta_{il})^{\beta}} \text{ Si } j \in J_i^k \\ p_{ij}^{k}(t) = 0 \qquad\qquad\qquad\qquad \text{Si } j \notin J_i^k \end{cases} \tag{3.3}$$

avec :

$\tau_{ij}(t)$: l'intensité de la piste sur la route *ij* à l' instant *t*

η_{ij} : la visibilité (désigne l'inverse de la distance séparant les positions *i* et *j*)

J_i^k : le voisinage de la fourmi *k*, c'est-à-dire l'ensemble des positions non encore visitées par la fourmi *k*

$\alpha \; et \; \beta$: Paramètres contrôlant l'importance de l'intensité de la piste et de la visibilité.

L'algorithme ACO est présenté par l'algorithme2. Il se compose d'une étape d'initialisation et d'une boucle comprend trois composants algorithmiques. Une itération simple de la boucle consiste à construire des solutions, ensuite les améliorer (composante facultative) avec un algorithme de recherche local et enfin mettre à jour les valeurs des phéromones.

Algorithme2 : Optimisation par Colonie de Fourmis
Initialisation des paramètres, les traînées de phéromone
Tant que (critère d'arrêt n'est pas atteint) **faire**
    Construire les solutions
    Appliquer une recherche locale (facultative)
    Mettre à jour phéromones
Fin tant que

Depuis l'apparition de l'ACO comme outil d'optimisation combinatoire, plusieurs tentatives ont été faites pour aborder les problèmes d'optimisation continus. Pour cela, des méthodes inspirées de l'ACO ont été proposées, tels que ACO_R (ant colony optimization R^n) par (Socha & Dorigo, 2008)[10].

L'idée fondamentale de l'ACO_R est d'employer une distribution de probabilité continue au lieu d'une distribution discrète.

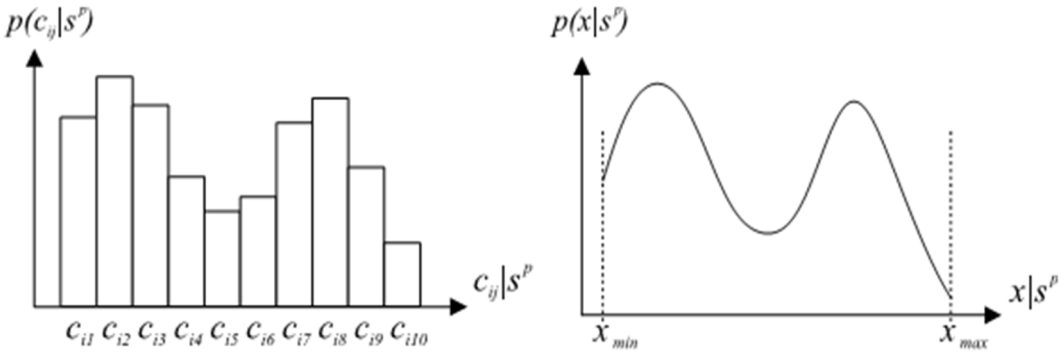


Figure.3.3 :a) Distribution discrète de probabilité b) Fonction de densité de probabilité(FDP)

Dans l'algorithme de colonie de fourmis pour les domaines continus, ACO_R, au lieu de choisir la route ij (c_{ij}) selon la formule (3.3), une fourmi échantillonne la fonction de densité de probabilité (FDP). L'une des fonctions les plus populaires qui est utilisée comme FDP facile à échantillonnée est la fonction gaussienne. Cependant, une fonction gaussienne simple ne peut pas décrire le cas où deux parties disjointes de l'espace de recherche sont attirantes, car elle a seulement un maximum. De ce fait, une FDP basée sur des fonctions gaussiennes est employée, mais légèrement différente, on utilise le noyau gaussien qui se définit comme une somme pondérée de plusieurs fonctions gaussiennes unidimensionnelles , $g_l^i(x)$, et désignées $G_i(x)$:

$$G^i(x) = \sum_{l=1}^{k} \omega_l g_l^i(x) = \sum_{l=1}^{k} \omega_l \frac{1}{\sigma_l^i \sqrt{2\pi}} e^{\frac{-(x-\mu_l^i)^2}{2\sigma_l^{i\,2}}} \qquad (3.4)$$

Où ω est le vecteur des poids liés aux différentes fonctions gaussiennes, μ^i est le vecteur des moyennes et σ^i est le vecteur des écarts type. Dans ACO_R, les solutions sont sauvegardées dans une archive appelée *archive de solution* (Figure.3.4). Au début de l'algorithme, l'archive est initialisée aléatoirement produisant *k* solutions. Les solutions $\{s_1,s_2,..s_l,..s_k\}$ avec leurs valeurs de fonction objective $\{f(s_1),f(s_2),..,f(s_l),..f(s_k)\}$ sont maintenues dans l'archive selon leur qualité. Dans un problème de minimisation on aura : $f(s_1) \leq f(s_2) \leq,..\leq f(s_l) \leq .. \leq f(s_k)$ et chaque solution s_l a un poids associé sur la base de la qualité de la solution, par conséquent: $\omega_1 \geq \omega_2 \geq \cdots \geq \omega_l \geq \cdots \geq \omega_k$

Le ω_l du poids de la solution s_l est calculée selon la formule (3.5):

$$\omega_l = \frac{1}{qk\sqrt{2\pi}} e^{\frac{-(l-1)^2}{2q^2k^2}} \tag{3.5}$$

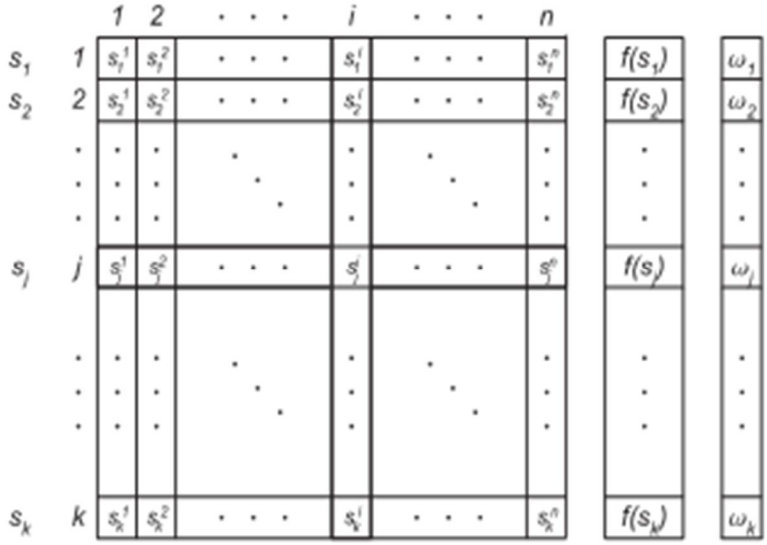


Figure.3.4 : Structure de l'archive des solutions

Selon (3.5), le poids est une valeur de la fonction gaussienne avec l'argument *l*, moyenne 1,0, et un écart type *qk*, où *q* est un paramètre de l'algorithme tel que lorsque *q* est faible, les solutions les mieux classés sont préférées. La solution qui a un rang plus élevé dans la table d'archive a plus de chance d'être choisie par les fourmis quand elles se déplacent vers une autre solution. L'écart type, entre les i^{iem} valeurs des variables de décision, doit être calculé en proportion avec la variable de décision donnée par (3.6).

À cet effet, par l'utilisation d'un générateur aléatoire normale, comme la méthode de Box-Muller, avec moyenne s_l^i et un écart-type σ_l^i, doit être produite pour la $i^{ème}$ variable de décision et ce processus continu jusqu'à la $n^{ième}$ variable de décision. Finalement, de nouvelles solutions, égal au nombre de fourmis, sont produits et ajoutés à la table d'archive. Après la collection de toutes les solutions, *k* solutions sont gardées et les autres sont supprimées. Et l'écart type mentionné est calculé comme suit:

$$\sigma_l^i = \varepsilon \sum_{e=1}^{k} \frac{\left|S_e^i - S_l^i\right|}{k-1} \tag{3.6}$$

Algorithmes de colonies de fourmis pour les domaines continues :

Construire la distribution de probabilité initiale
Tant que (critère d'arrêt non atteint)
Pour chaque fourmi, de a=1 à m
Pour chaque variable, de i =1 à n:
Choisir aléatoirement une valeur x_i selon la distribution $P_i(x_i)$
Ajouter à la solution en construction
Fin
Fin
Mémoriser les k meilleures solutions trouvées
Reconstruire la distribution de probabilité selon les meilleures solutions
Fin

3.4. Algorithme de recherche gravitationnel (GSA)

L'optimisation par *algorithme de recherche gravitationnel* est une nouvelle métaheuristique d'optimisation inspirée de la nature, développée par Rashedi et al en 2009 [42]. Cet algorithme est basé sur la loi de gravité de Newton qui décrit la gravitation comme une attraction entre des corps ayant une masse : « Deux particules dans l'univers s'attirent avec des forces qui sont directement proportionnelles au produit des masses des particules et inversement proportionnelles au carré de la distance qui les sépare ».

Dans le GSA, les masses des objets (solutions) sont proportionnelles à leurs valeurs de fonctions objectives (coûts). A chaque itération, les masses s'attirent, entre elles, par les forces de gravitation. La masse, la plus lourde à la force d'attraction la plus grande. Par conséquent, les masses plus lourdes qui sont probablement près de l'optimum global attirent les autres masses selon leurs distances. Chaque objet est déterminé par quatre spécifications : position, inertie, masse gravitationnelle active et la masse gravitationnelle passive. La position correspond à une solution du problème, l'inertie et les masses gravitationnelles sont déterminées en utilisant la fonction objective. Plus précisément, les masses obéissent les deux lois suivantes:

- Loi de gravité : chaque objet attire un autre par une force gravitationnelle directement proportionnelle au produit de leurs masses et inversement proportionnelle à la distance entre eux.

- Loi de mouvement: la vitesse courante de toute masse est la somme de la fraction de sa vitesse précédente et la variation de la vitesse.

Dans le GSA, les agents sont considérés comme des objets qui s'attirent entre eux engendrant un mouvement global pour atteindre les objets les plus lourds. La position du i^{iem} objet est définit par (3.7) :

$$X_i = (x_i^1, \dots, x_i^d, \dots, x_i^n) \quad \text{pour } i = 1,2, \dots, N \tag{3.7}$$

Où x_i^d montre la position d'i^{iem}objet selon la d^{iem} dimension.

La force de gravitation de *j* qu'influe sur *i* est donnée par (3.8) :

$$F_{ij}^d(t) = G(t)\frac{M_{pi}(t)\times M_{aj}(t)}{R_{ij}(t)+\varepsilon}(x_j^d(t) - x_i^d(t)) \tag{3.8}$$

Avec :

M_{aj} : est la masse de gravité active liée à l'agent *j*

M_{pi} : est la masse de gravité passive liée à l'agent *i*

$G(t)$: constante de gravité à l'instant *t*, ε : petit constant.

$$R_{ij}(t) = \|X_i(t), X_j(t)\|_2 \tag{3.9}$$

est la distance euclidienne entre l'objet *i* et l'objet *j*

Pour donner un caractère stochastique au GSA, la force totale qui agit sur l'agent *i* dans la dimension *d* est donnée par (3.10) :

$$F_i^d(t) = \sum_{j=1, j\neq i}^{N} rand_j F_{ij}^d(t) \tag{3.10}$$

Avec $rand_j$ est un nombre aléatoire entre 0 et 1.

Selon la loi du mouvement, l'accélération d'un objet est proportionnelle à la force résultante et inversement proportionnelle à sa masse. L'accélération de tous les objets devrait être calculée selon l'équation (3.11) :

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)} \tag{3.11}$$

Avec M_{ii} est l'inertie de l' i^{iem}objet.

La vitesse et la position des objets sont calculées selon les équations (3.12) et (3.13) :

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t) \tag{3.12}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \tag{3.13}$$

G est en fonction de la valeur initial (G_0) et le temps t, elle est initialisée au début ensuite elle est diminuée avec le temps pour contrôler la précision de la recherche. Les inerties et les masses gravitationnelles sont actualisées par les équations suivantes :

$$M_{ai} = M_{pi} = M_{ii} = M_i, \quad i = 1,2,\dots,N \tag{3.14}$$

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)} \tag{3.15}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)} \tag{3.16}$$

Avec :

$fit_i(t)$: représente la valeur de la fonction objective de l'objet *i* à l'instant *t*

$$worst(t) = max_{j\in\{1,\dots,N\}} fit_j(t) \tag{3.17}$$

$$best(t) = min_{j\in\{1,\dots,N\}} fit_j(t) \tag{3.18}$$

L'algorithme GSA est défini par les étapes suivantes :

a) Identification de l'espace de recherche.

b) initialisation aléatoire.

c) Évaluation de la fonction objective des objets.

d) Mise à jour *G (t),best(t), worst(t)* et *Mi* pour i = 1,2, ..., *N*.

e) Calcul de la force total dans les différentes directions.

f) Calcul de l'accélération et la vitesse.

g) mise à jour des positions des objets.

h) Répétez les étapes c à g jusqu'à ce que le critère d'arrêt est atteint.

i) Fin

3.5.Application

Dans cette section, nous appliquons et nous comparons les trois métaheuristiques ci-dessus pour la solution de la commande prédictive linéaire et non-linéaire. Le temps de calcul nécessaire pour obtenir la solution de la commande prédictive est enregistré (à chaque période d'échantillonnage). Le temps de calcul est atteint utilisant un ordinateur de type « Intel®

Pentium Core i3 » à 3.3 GHz avec 4Go de RAM.

Pour la commande des systèmes linéaires, la fonction coût à minimiser est donnée par la formule (3.19)

$$\min_{u}\{ J(U, x(t)) = x_{t+N/t}^{T} P x_{t+N/t} + \sum_{k=0}^{N-1} x_{t+k/t}^{T} Q x_{t+k/t} + u_{t+k}^{T} R u_{t+k} \} \quad (3.19)$$

La formule (3.20) présente la fonction coût qu'est utilisée pour la commande du système non-linéaire (robot mobile)

$$J_{N_p}(x, k, \mathcal{U}) = F\left(x(k + N_p)\right) + \sum_{i=k}^{k+N_p-1} L(x(i), u(i)) \quad (3.20)$$

Dans la partie suivante, on va présenter quelques résultats (simulation et expérimental) pour l'utilisation des métaheuristiques, décrites précédemment, pour la commande des différents systèmes par la MPC.

3.6. Résultats de simulation:

3.6.1. Systèmes linéaires :

3.6.1.1. Système mono variable

On considère le système linéaire sous contraints décrit par l'équation d'état suivante [5] :

$$x(t+1) = \begin{bmatrix} 0.7326 & -0.0861 \\ 0.1722 & 0.9909 \end{bmatrix} + \begin{bmatrix} 0.0609 \\ 0.0064 \end{bmatrix} u(t) \quad (3.21)$$

$$y(t) = [0 \quad 1.4142] x(t)$$

Tel que $-2 \leq u(t) \leq 2$

Avec $Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $R = 0.01$, et N=2.

Le problème est un problème de régulation vers l'origine, il consiste à minimiser la fonction objectif donnée par (3.19).

La matrice de coût final *P* est calculée à partir de l'équation algébrique de *Riccati* avec l'hypothèse que les contraintes ne sont pas actives pour k ≥ N. La période d'échantillonnage est *T=0.1s,* et les conditions initiales sont *x(0)* = [1 1]'.

On construit un contrôleur basé sur le problème d'optimisation (3.19), utilisant les approches présentées précédemment. Les solutions qui violent les contraintes sont pénalisées en ajoutant une valeur élevée à la fonction coût. Les figures 3.5 et 3.6 représentent les résultats de simulations. Le signal de commande et les états sont identiques pour les trois heuristiques. Ces résultats sont semblables avec ceux obtenus dans [5] utilisant une programmation multi paramétrique. Le tableau 3.1 donne les temps de calcul où on remarque que l'algorithme PSO converge plus rapidement que les autres.

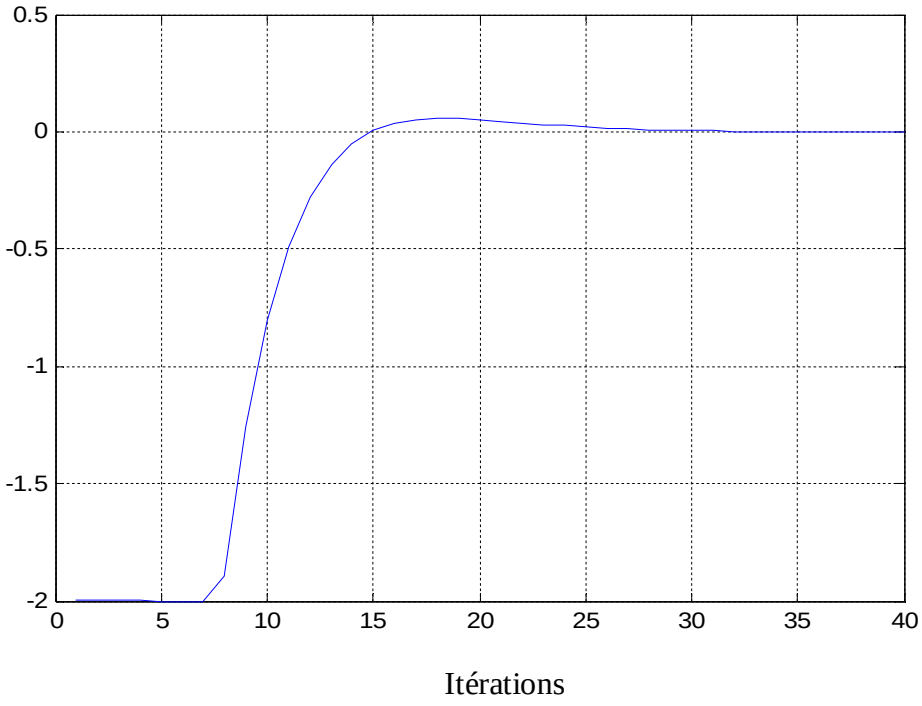


Figure.3.5 : Signal de commande pour l'exemple1, PSO, ACO et GSA

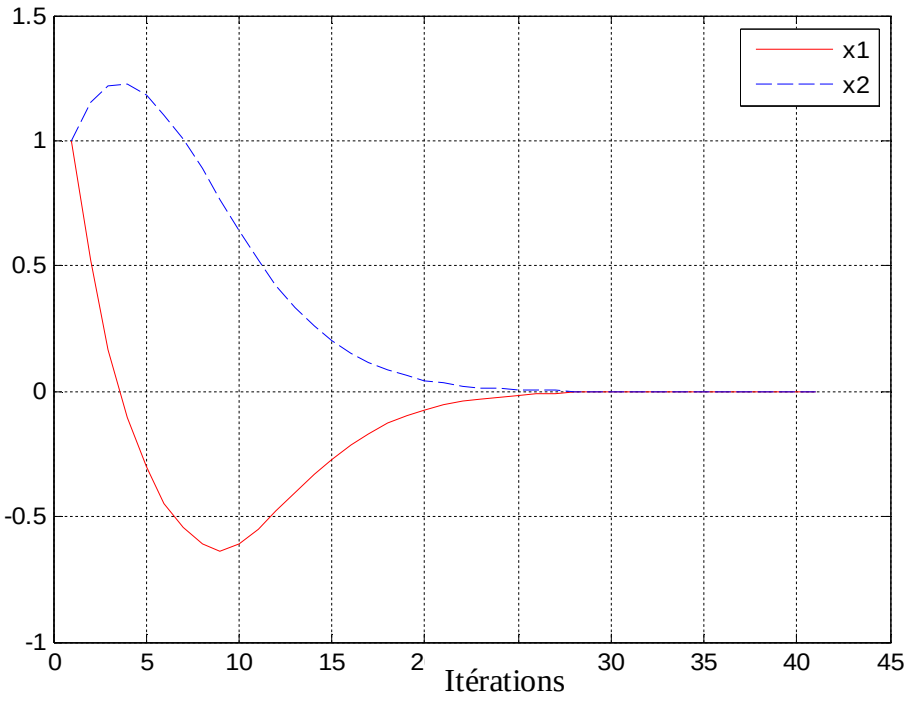


Figure.3.6 : Etats pour exemple1, PSO, ACO et GSA

	PSO	**ACO**	**GSA**
Temps de calcul	≤ 1.5ms	≤ 5ms	≤ 3.2ms

Tableau3.1 : Temps de calcul

3.6.1.2. Système multi variables

Dans ce second exemple, on considère un modèle d'un hélicoptère de laboratoire (Quanser 3-DOF hélicoptère). La période d'échantillonnage est $T = 0.01\ s$. La représentation du système échantillonné dans l'espace d'état est donnée [43]:

$$A = \begin{bmatrix} 1 & 0 & 0.01 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0.01 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0.01 & 0 & 0 & 0 & 1 \end{bmatrix};\ B = \begin{bmatrix} 0 & 0 \\ 0.0001 & -0.0001 \\ 0.0019 & 0.0019 \\ 0.0132 & -0.0132 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (3.22)$$

Les états du système sont:

x1: élévation,

x2: angle de tangage,

x3: taux d'élévation,

x4 : taux d'angle d'inclinaison,

x5: intégrale de l'erreur d'altitude,

x6: intégrale de l'erreur de l'angle de tangage.

Les entrées du système sont:

u1: puissance du rotor du front,

u2: puissance du rotor arrière.

Le but est de conduire le système vers l'origine avec les contraintes sur les entrées et les états suivantes:

$-1 \le u1 \le 3;$

$-1 \le u2 \le 3;$

$-0.44 \le x3 \le 0.44;$

$-0.6 \le x4 \le 0.6;$

Comme dans l'exemple 1, on minimise une fonction coût sous la forme de (3.19) avec :

Q = *diag*(100; 100; 10; 10; 400; 200); *R* = I2×2, *P* est calculée à partir de l'équation algébrique de *Riccati* avec l'hypothèse que les contraintes ne sont pas actives pour k ≥ N. L'horizon de prédiction est *Np=3*, et l'horizon de commande est *Nu=2*.

Les résultats de cette simulation sont présentés dans les Figures 3.7 à 3.10 où on peut remarquer que le signal de commande et les états sont semblables pour le PSO et ACO, mais légèrement différente de celle de GSA, car ce dernier n'a pas atteint le minimum global. Aussi, ces résultats sont très similaires à ceux obtenus dans [43] utilisant la programmation quadratique multi paramétrique. Le tableau 3.2 donne le temps de calcul où l'on peut remarquer que l'algorithme PSO est le plus rapide.

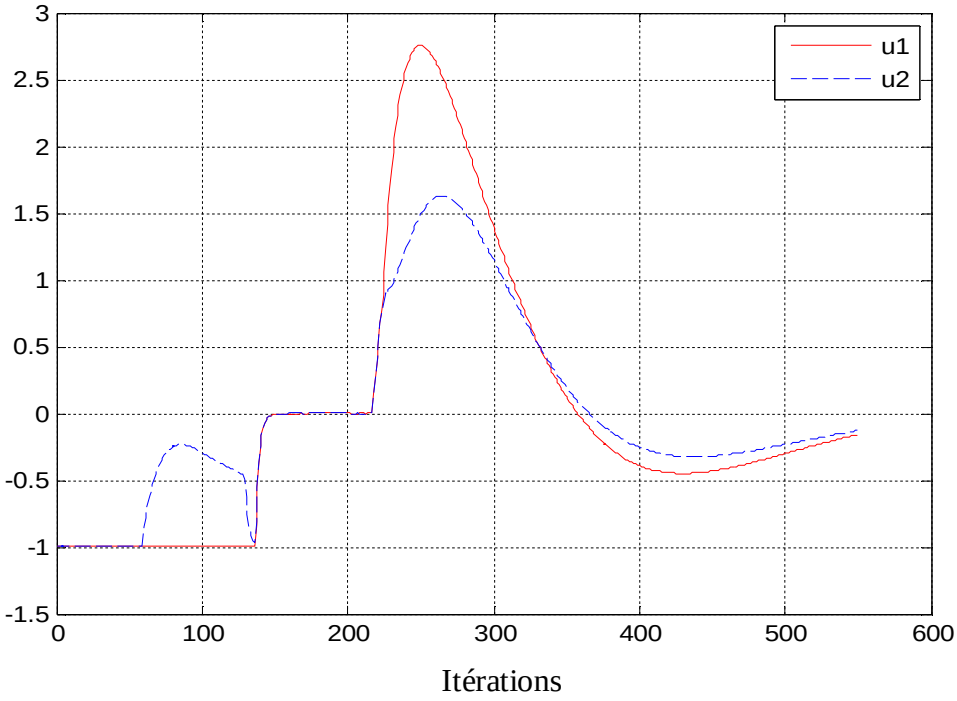


Figure.3.7: Le signal de commande pour l'exemple 2, PSO et ACO

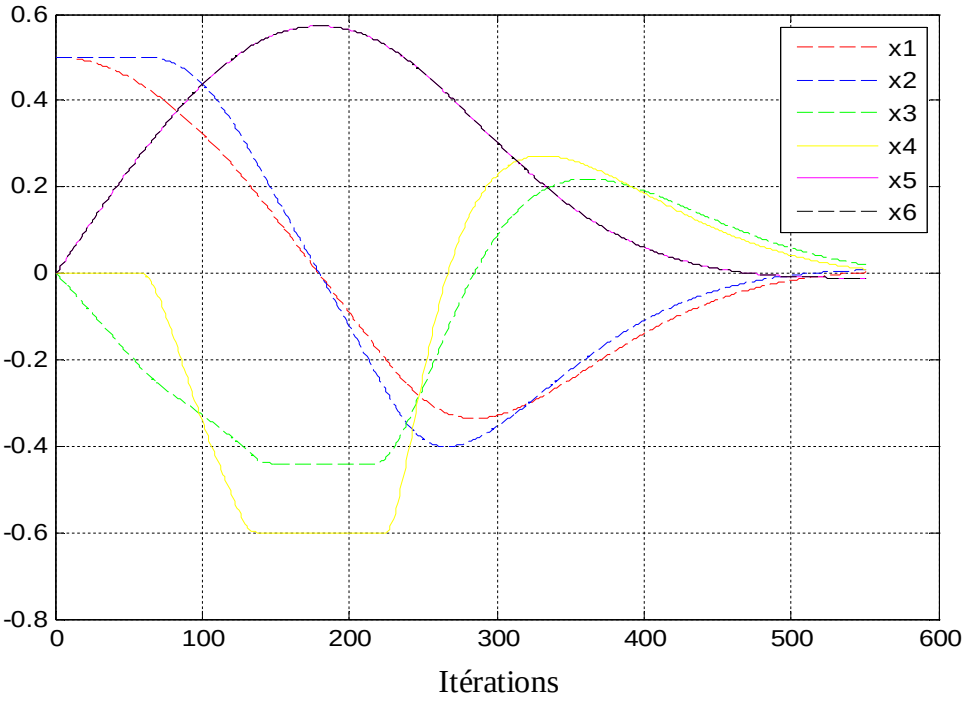


Figure.3.8 : Etats pour exemple 2, PSO et ACO

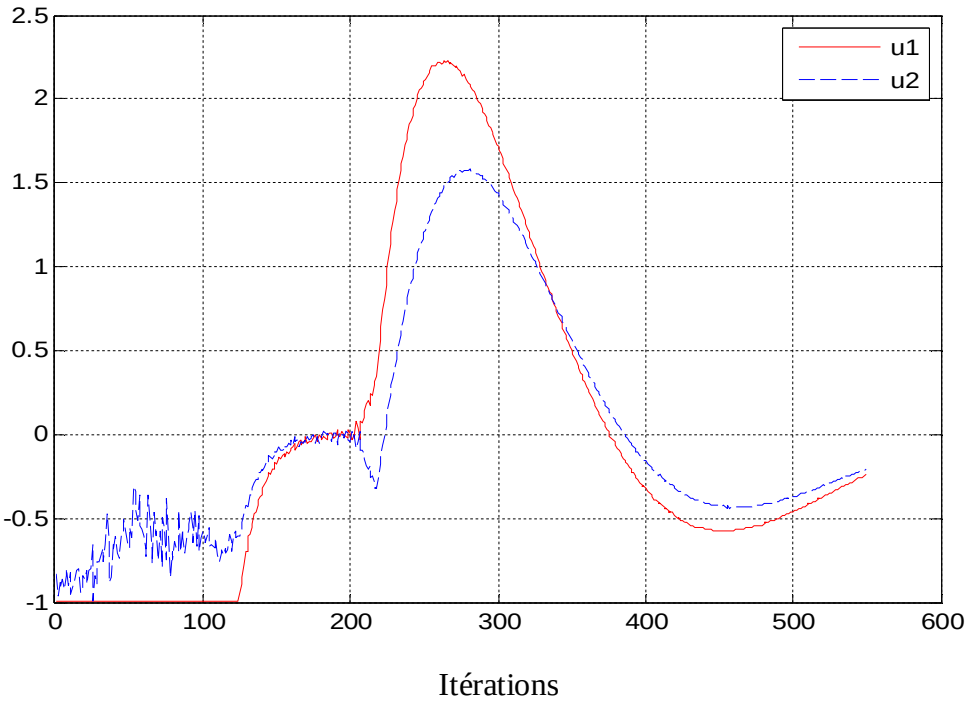


Figure.3.9: Le signal de commande pour l'exemple 2, GSA

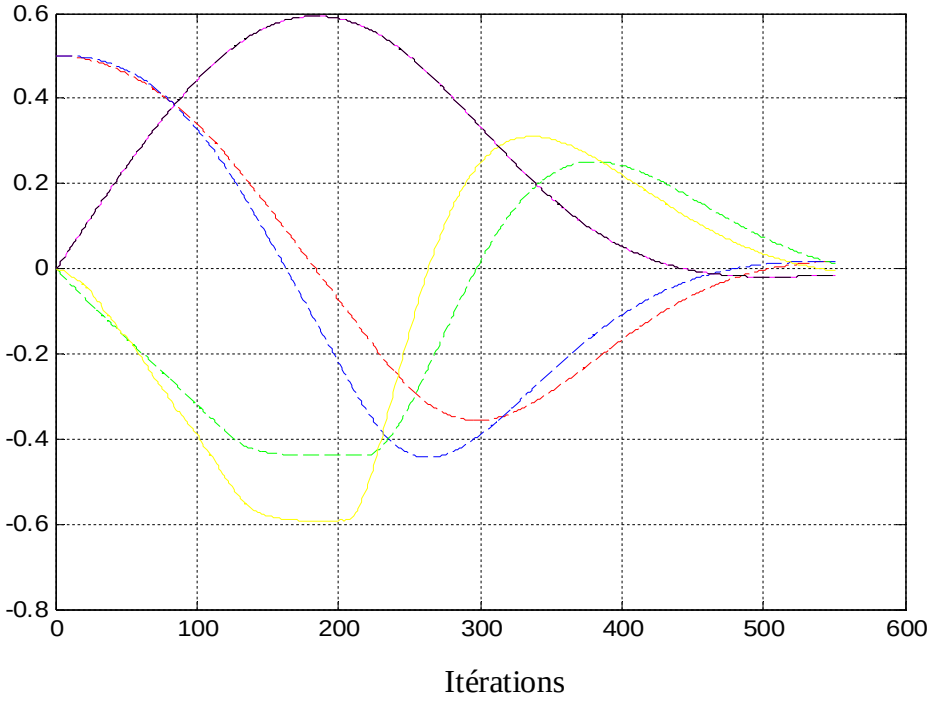


Figure.3.10 : Etats pour exemple2, GSA

	PSO	ACO	GSA
Temps de calcul	≤ 7.5ms	≤ 85ms	≤ 55ms

Tableau 3.2 : Temps de calcul pour l'hélicoptère

3.6.2. Systèmes non-linéaires :

Dans cet exemple, nous comparons l'application des trois métaheuristiques présentées précédemment pour la commande d'un système non-linéaire multi variable, un robot mobile à deux roues. Ce système est montré dans la figure 3.11, où (*x*, y) est la position du centre de l'axe des roues *C et* θ est l'orientation de robot.

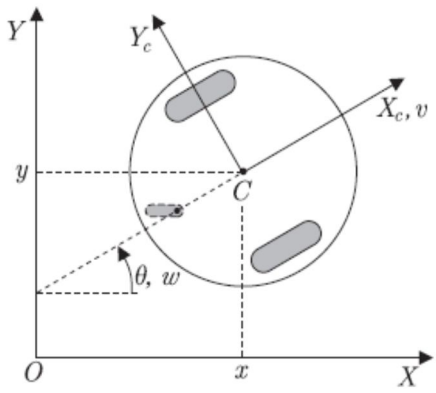


Figure.3.11 : Robot mobile utilisé

Le modèle cinématique du ce robot est donné par :

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\cos\theta & 0\\ \sin\theta & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix} \tag{3.23}$$

v est la vitesse de translation et ω est la vitesse angulaire.

Le problème est de trouver la loi de commande définie par $v(t)$ et $\omega(t)$ qui permet au robot de suivre une trajectoire de référence définie par : $[x_r(t) y_r(t)]$ utilisant la commande prédictive non-linéaire. Le temps de calcul pour obtenir la solution de la commande prédictive non-linéaire est enregistré. Dans ce travail on suppose qu'on a toutes les informations sur l'espace de navigation du robot, à chaque instant *t*. La trajectoire de référence est donnée par :

$$x_r(t) = cos(\omega_0 t)\,;\; y_r(t) = sin(2 * \omega_0 t) \tag{3.24}$$

Le problème d'optimisation de la commande prédictive est donné par

$$minJ = min(\sum_{k=1}^{N}(x(t+k) - \cos\omega_0(t+k))^2 + ((y(t+k) - \cos\omega_0(t+k))^2) + r_1 v^2(t+k) + r_2\omega^2(t+k)) \tag{3.25}$$

Avec les contraintes sur le signal de commande :

$-0.47 \leq v \leq 0.47$;

$-3.77 \leq \omega \leq 3.77$;

Ce problème d'optimisation non-convexe est résolu à chaque période d'échantillonnage pour obtenir une séquence de $v(t+k)$et une séquence de $\omega(t+k)$, *k=1…N,* mais seulement $v(t)$ et $\omega(t)$ sont appliquées au robot.

La satisfaction des contraintes est assurée par l'ajout d'un coût élevé à la fonction objective. Les variables du problème d'optimisation sont : coordonnées de la particule dans le PSO, coordonnées de la masse dans le GSA et coordonnées de la fourmi dans l'ACO. La dimension du problème d'optimisation est $2 \times N$. L'algorithme de la commande prédictive a été exécuté avec un horizon *N*=3, $r_1 = 1$ et $r_2 = 1$. Les paramètres des métaheuristiques sont donnés au tableau 3.3. La convergence des algorithmes vers une bonne solution après un certain nombre d'itérations présentée dans le même tableau.

Paramètre du PSO	Taille de l'essaim=15, Inertie=1, C1=C2=1.2, Itérations=25
Paramètre de l'ACO	Nombre de fourmis=2, K=42, q=0.0001,ε = 0.85, Itérations=50
Paramètre du GSA	Nombre d'agent=8, G0=1,α = 20, ε = 0.1, Itérations=1000

Tableau 3.3 : Paramètres des métaheuristiques

Trois scénarios sont considérés :

- Suivi de trajectoire sans obstacles
- Suivi de trajectoire avec obstacles fixes
- Suivi de trajectoire avec obstacles fixes et dynamiques

3.6.2.1. Scénario N°01 : Suivi de trajectoire sans obstacles

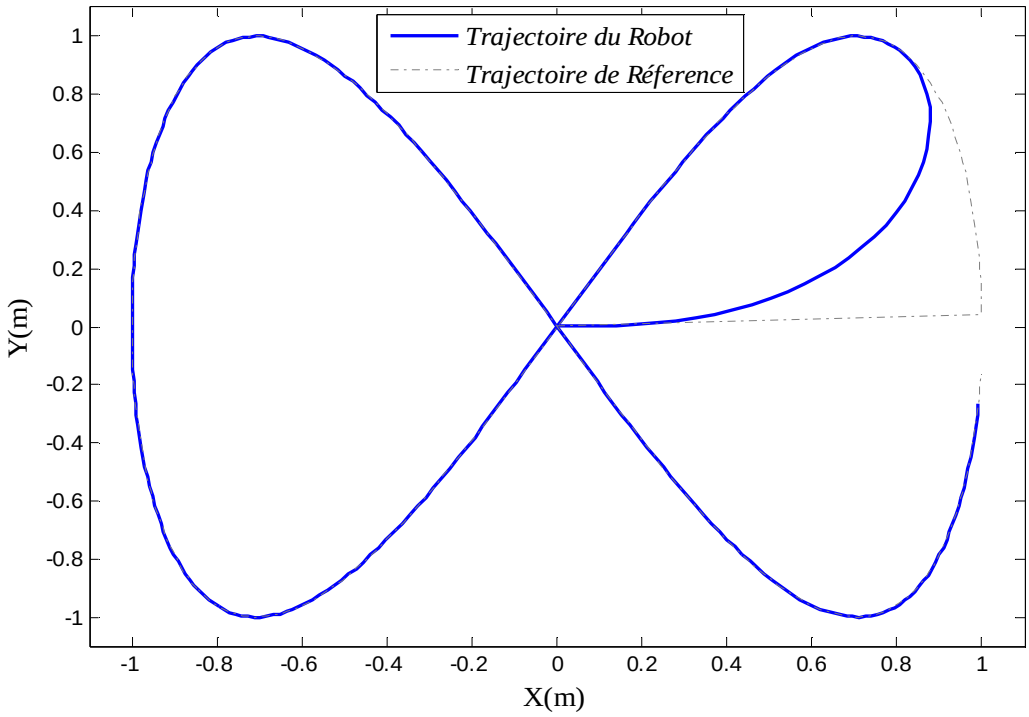


Figure.3.12 : Trajectoire du robot : MBPC avec PSO et ACO

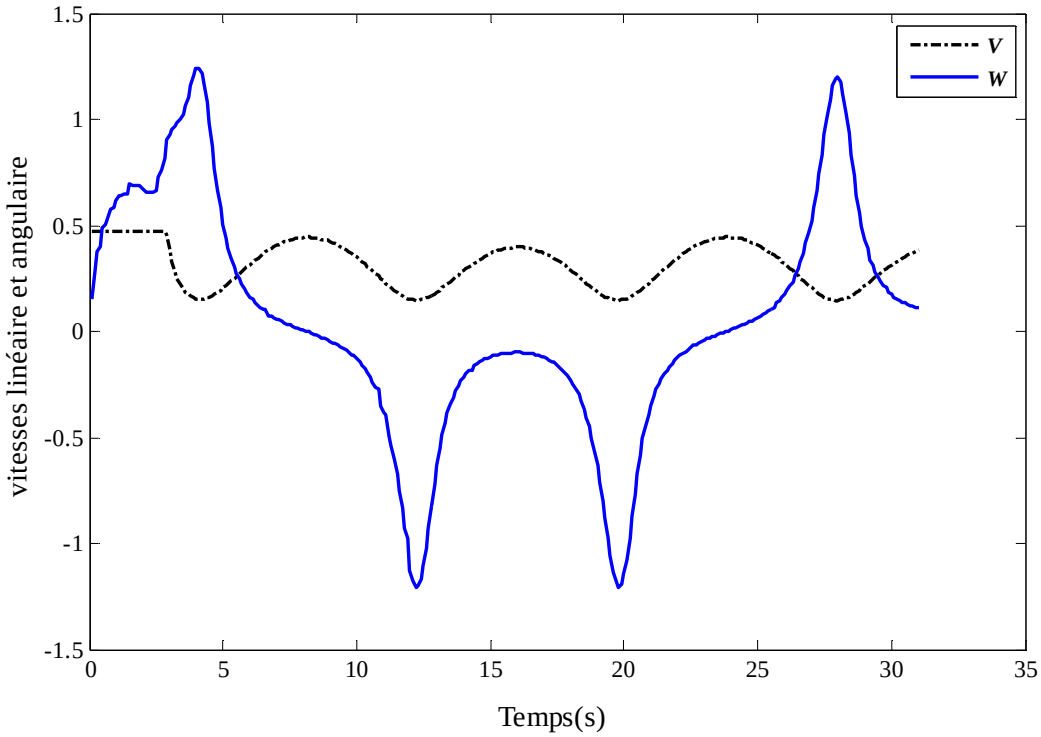


Figure.3.13 : Les signaux de commande:MBPC avec le PSO et ACO

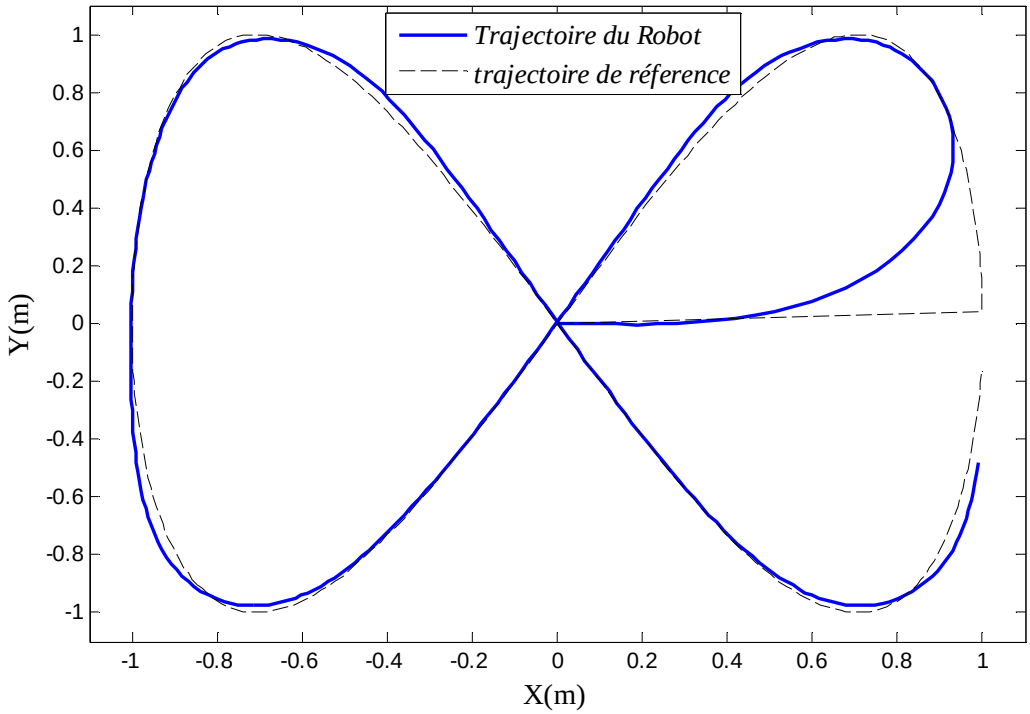


Figure.3.14 : Trajectoire du robot : MBPC avec GSA

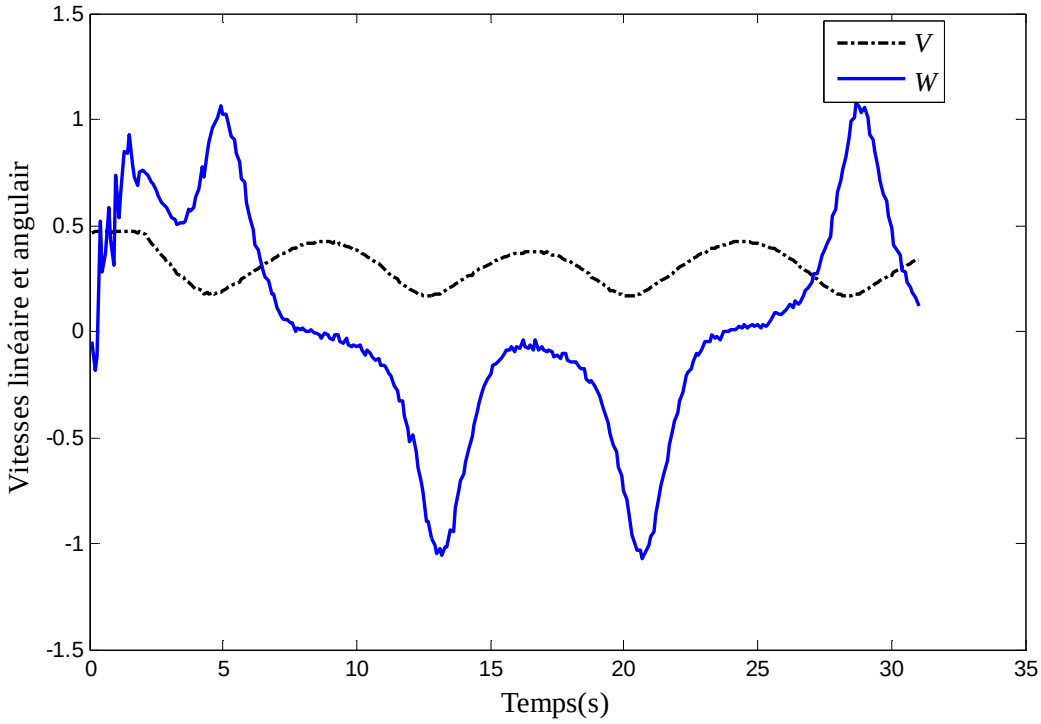


Figure.3.15: Les signaux de commande:MBPC avec le GSA

	PSO-NMPC	**ACO-NMPC**	**GSA-NMPC**
Temps de calcul	$\leq$ 4ms	$\leq$ 40ms	$\leq$ 30ms

Tableau3.4 : Temps de calcul pour le 1^{ier} scénario

Les résultats du premier scénario sont montrés sur les figures 3.12 et 3.13 pour le PSO et ACO et sur les figure3.14 et 3.15 pour GSA. Les temps de calcul sont indiqués dans le tableau3.4, où on peut remarquer que le PSO est plus performant que les deux autres algorithmes. Le GSA n'a pas pu produire un bon suivi alors que l'ACO produit un suivi semblable au PSO mais après un temps de calcul plus long. Les contraintes sur les signaux de commande sont toujours satisfaites comme il est montré sur la figure 3.13 et la figure3.15. Pour le GSA, le signal de commande est actif et n'a pas pu être amélioré même après un temps de calcul supérieur à 30ms. Il semble que l'algorithme soit tombé dans un « mauvais » minimum local.

3.6.2.2. Scénario N°2 : Suivi de trajectoire avec obstacles fixes

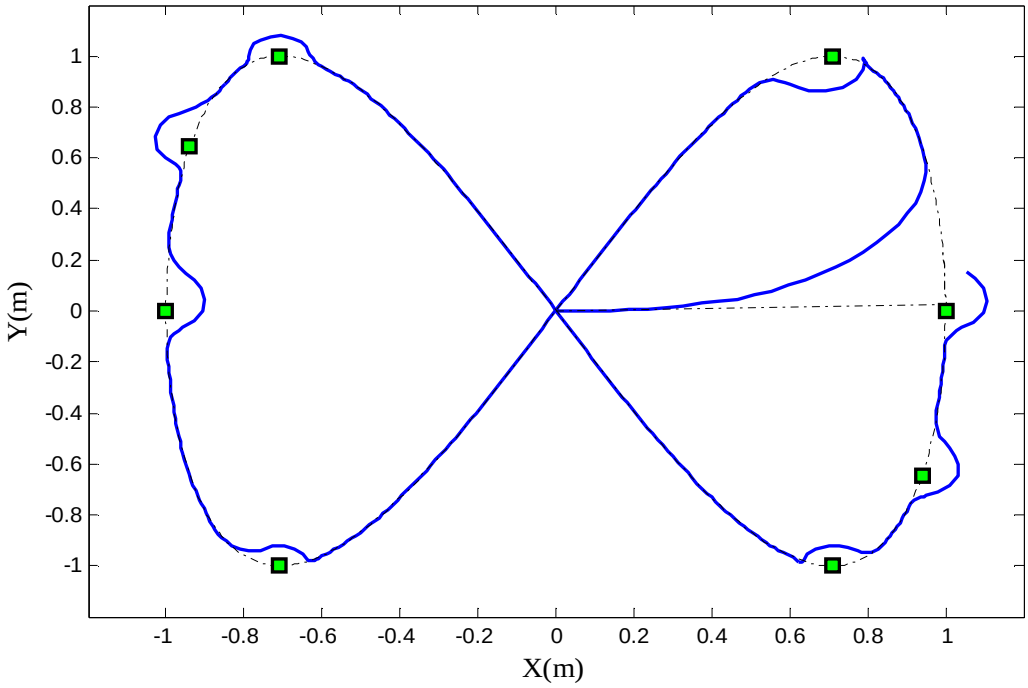


Figure.3.16: Trajectoire du robot pour l'évitement d'obstacles fixes: MBPC avec PSO

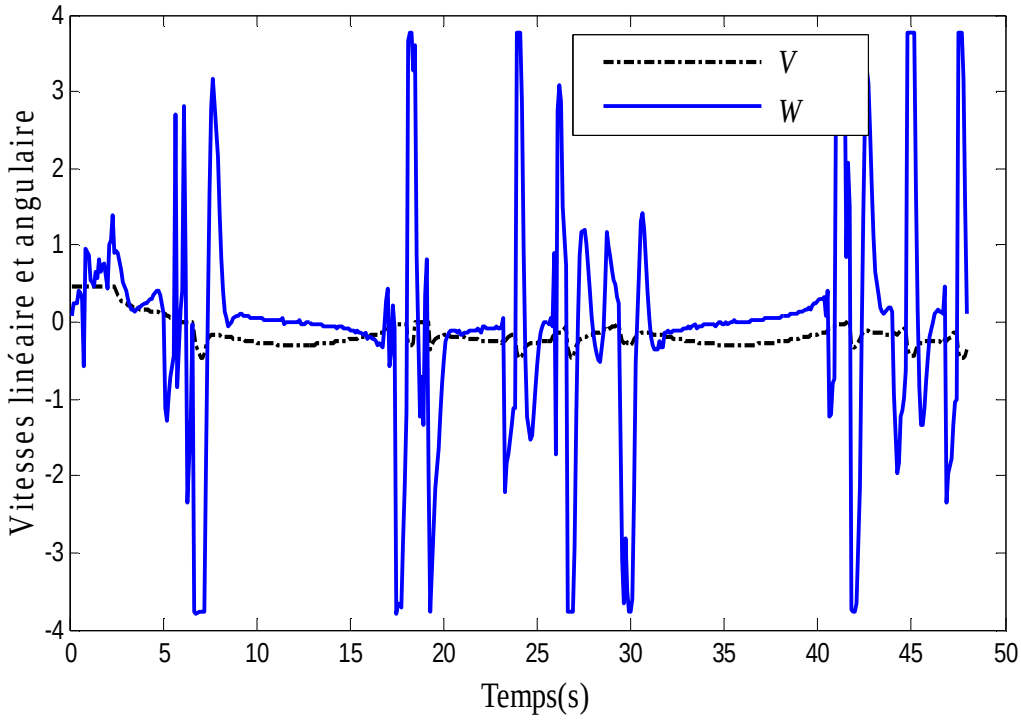


Figure.3.17 : Les signaux de commande : MBPC avec PSO

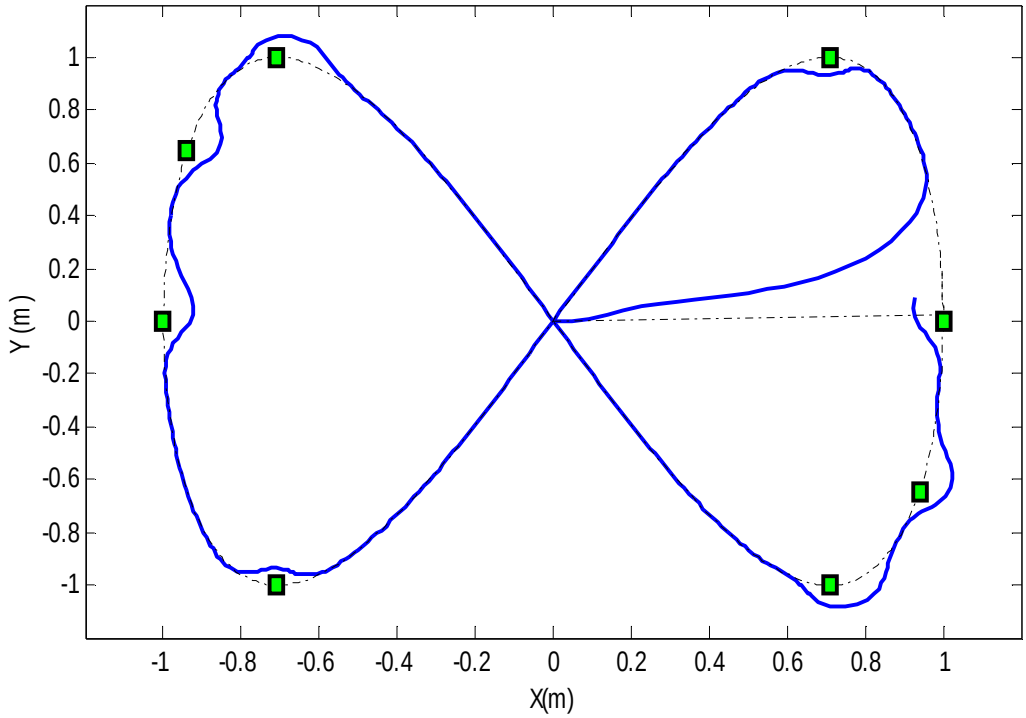


Figure.3.18 : Trajectoire du robot pour l'évitement d'obstacles : MBPC avec ACO

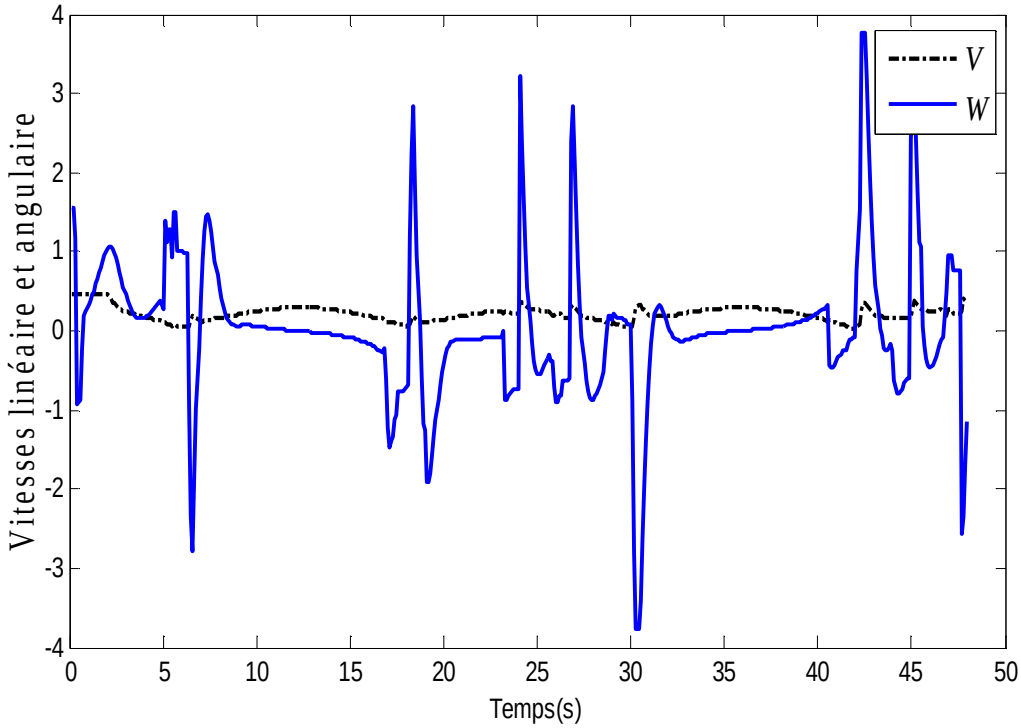


Figure.3.19 : Les signaux de commande : MBPC avec ACO

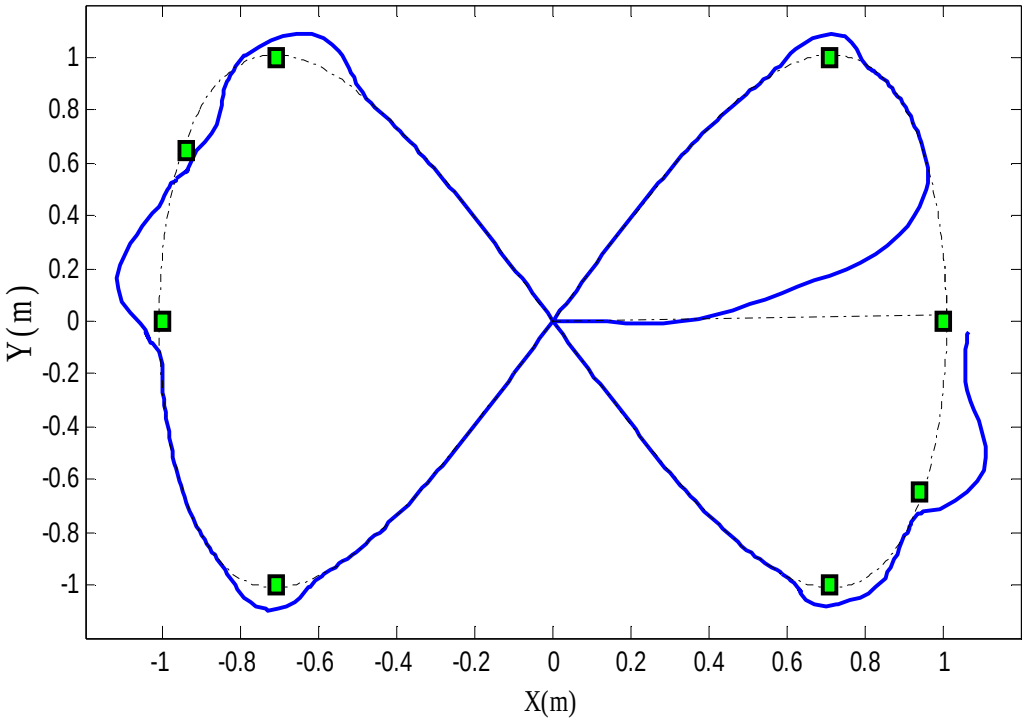


Figure.3.20 : Trajectoire du robot pour l'évitement d'obstacles : MBPC avec GSA

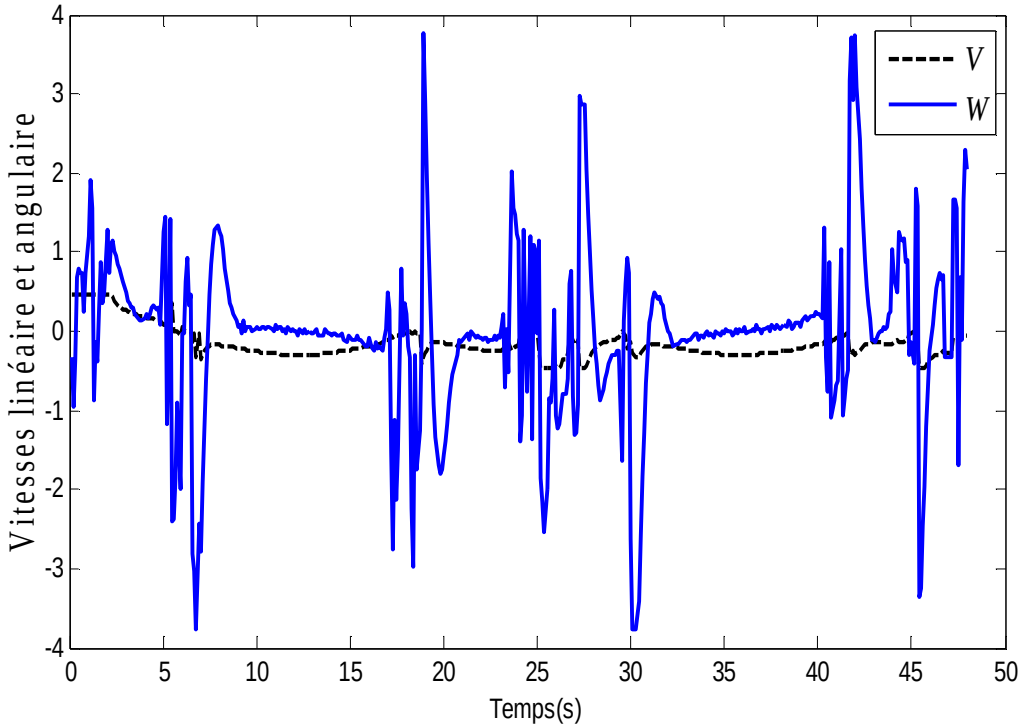


Figure.3.21 : Les signaux de commande : MBPC avec GSA

	PSO NMPC	ACO-NMPC	GSA-NMPC
Temps de calcul	≤ 4ms	≤ 47ms	≤ 32ms

Tableau3.5 : Temps de calcul pour le 2^{iem} scénario

Dans le deuxième scénario, l'évitement d'obstacle est assuré en ajoutant une contrainte au problème de la MBPC : si la distance entre l'obstacle et le robot est inferieure à une valeur indiquée *r*, une pénalité est ajoutée à la fonction coût.

Les résultats de ce scénario sont montrés sur les figures de 3.16 à 3.21 pour PSO, ACO et GSA respectivement. Les temps de calcul sont présentés dans le tableau3.5, où on peut remarquer que le PSO produit de meilleurs résultats que les deux autres algorithmes dans l'évitement d'obstacle et la qualité des signaux de commande. La solution obtenue avec le GSA produit un évitement loin de l'obstacle produisant mauvais suivi. Les contraintes sur le signal de commande sont toujours satisfaites.

3.6.2.3. Scénario N°03 : Suivi de trajectoire avec obstacles fixes et dynamiques

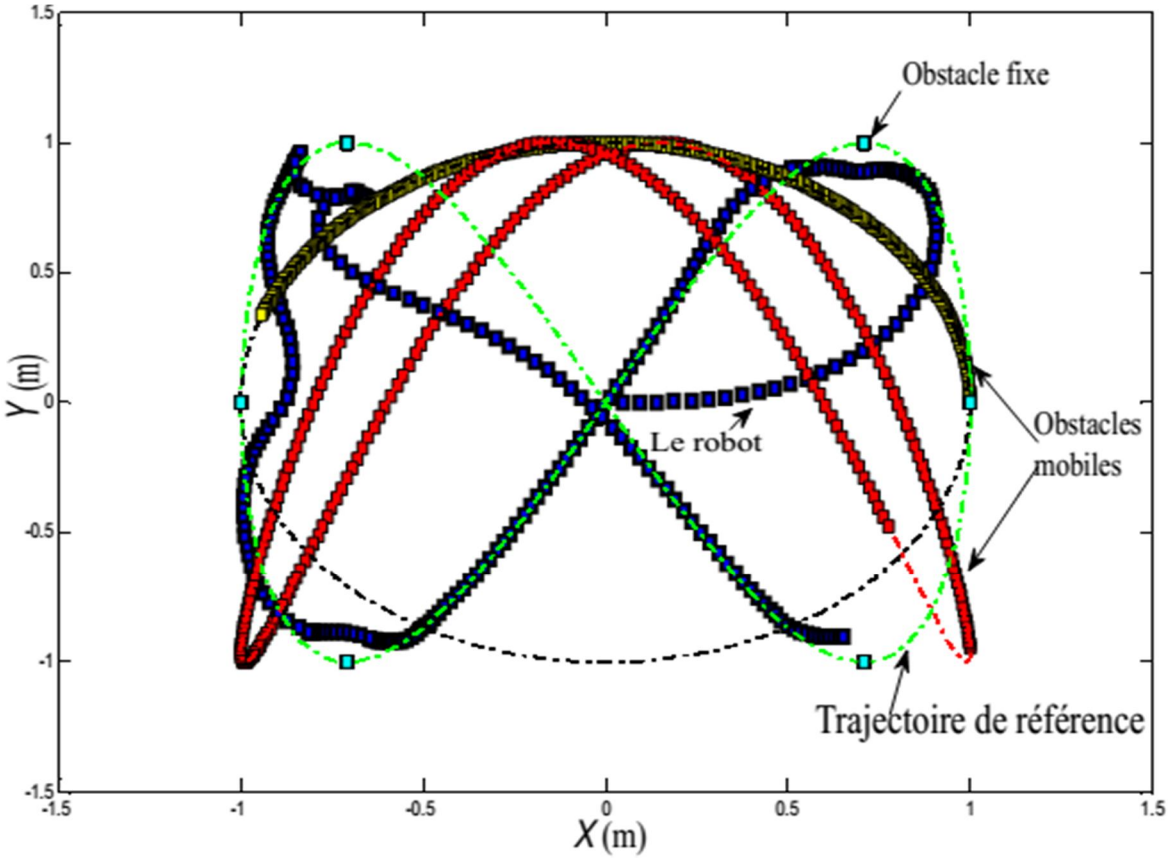


Figure.3.22 : Résultats du troisième scénario: La trajectoire du robot et les obstacles mobiles

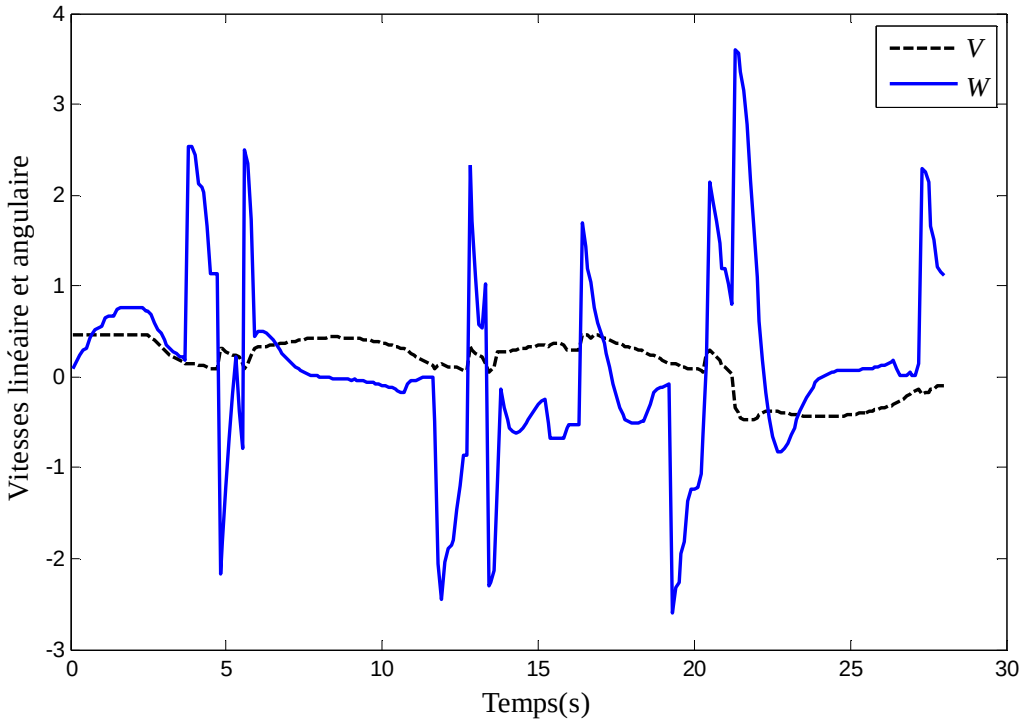


Figure.3.23 : Les signaux de commande

	PSO NMPC	**ACO-NMPC**	**GSA-NMPC**
Temps de calcul	≤ 4ms	≤ 55ms	≤ 40ms

Tableau3.6 : Temps de calcul pour le 3^{iem} scénario

Comme dans le deuxième scénario, l'évitement d'obstacles fixes et dynamiques est assuré par l'ajout d'une contrainte au problème de la MBPC. Les résultats de ce scénario sont montrés sur les figures 3.22 et 3.23 pour la trajectoire du robot et les signaux de commande. L'algorithme de la MPC à base du PSO a été simulé le premier et sa solution a été enregistrée. Les deux autres algorithmes ont été simulés après jusqu'à ce qu'une solution semblable à celle du PSO soit obtenue. Les temps de calcul sont donnés au tableau 3.6, où le PSO est encore beaucoup plus rapide que l'ACO et le GSA.

Les résultats de simulation montrent que l'algorithme d'optimisation par essaim de particules peut être considéré comme une alternative pour les applications temps réel qui ont une période d'échantillonnage de l'ordre de quelque milliseconde. Dans la partie suivante, la MPC

non-linéaire à base de l'algorithme d'optimisation par essaim de particules est appliquée pour la commande d'un robot mobile pour le suivi de trajectoire et l'évitement d'obstacles.

3.7. Résultats expérimentaux :

3.7.1. Le robot LEGO :

Lego Mindstorms NXT est un jouet relativement sophistiqué de construction et de robotique présenté par LEGO en 2006. Les robots Lego Mindstorms NXT sont très utilisés pour l'enseignement et l'éducation des sciences et techniques. Les caractéristiques importantes de la gamme Lego Mindstorms NXT sont :

- brique NXT intelligente programmable : elle représente le centre de commande et énergétique du robot. Elle est composée essentiellement de quatre ports d'entrée, 3 ports de sortie de similaire au RJ 12, un écran et des boutons permettent de naviguer dans le contenu de la brique. Elle est à base d'un microprocesseur 32 bit ARM7 d' Atmel. Elle supporte la prise en charge de communications sans fil Bluetooth et s'alimente au moyen de 6 piles AA de 1,5 V, ou d'une batterie rechargeable.

Figure.3.24 : La brique Mindstorms NXT

- des servomoteurs : un robot Lego Mindstorms NXT, dans sa configuration de base, est équipé de trois servomoteurs dont deux sont généralement utilisés pour la mobilité du robot (le faire avancer, reculer ou tourner), le troisième moteur est exploité pour la mobilité d'un organe terminal, par exemple la fermeture d'une pince, porte un capteur, etc. Il tourne avec une résolution de rotation d'un degré pour

commande précise et envoi les rapports à la brique intelligente NXT. Il fonctionne à 177 t/mn, avec un couple de 16.7 N.cm.

Figure.3.25 : Servomoteur Mindstorms NXT

Il existe de nombreuses possibilités pour programmer le Mindstorms NXT. Le NXT-G est un logiciel de programmation graphique fourni dans le kit grand public. Dans ce travail, on a utilisé le logiciel LABVIEW pour commander le robot Lego Mindstorms NXT.

3.7.2. Modèle cinématique du robot mobile utilisé:

Le modèle cinématique du robot réel est donné par :

$$\dot{x}(t) = \frac{v_d(t)+v_g(t)}{2} cos\theta(t) \tag{3.26}$$

$$\dot{y}(t) = \frac{v_d(t)+v_g(t)}{2} sin\theta(t) \tag{3.27}$$

$$\dot{\theta}(t) = \frac{v_d(t)-v_g(t)}{b} \tag{3.28}$$

$v_d \in R$ et $v_g \in R$ sont les vitesses linéaires de la roue droite et la roue gauche, respectivement, $b \in R$ est la distance entre les centres des roues et θ est l'angle d'orientation du robot.

Le problème est de trouver la loi de commande définie par $v_d(t)$ et $v_g(t)$ qui permet au robot de suivre une trajectoire de référence définie par :$[x_r(t) y_r(t)]$ utilisant la commande prédictive non-linéaire.

3.7.3. Commande du robot LEGO

3.7.3.1. Scénario 01 : suivi d'une trajectoire circulaire :

La trajectoire de référence est donnée par :

$x_r(t) = cos(\omega_0 t)$; $y_r(t) = sin(\omega_0 t)$;$\omega_{0=} 0.15rad/s$

Les résultats du premier scenario sont présentés dans la figure 3.27:

Figure.3.26: Le robot Lego Mindstorms NXT

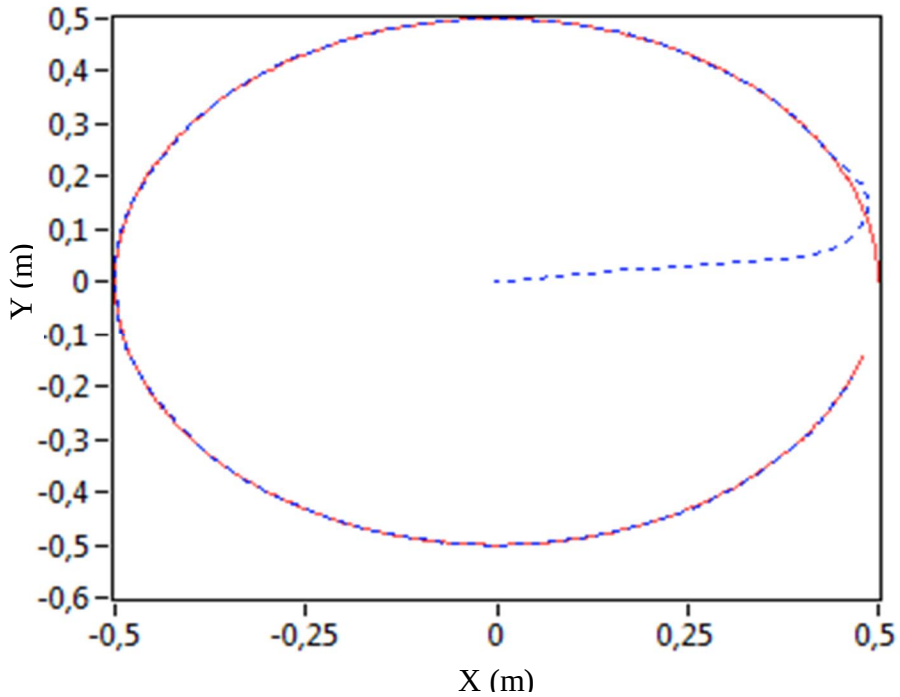
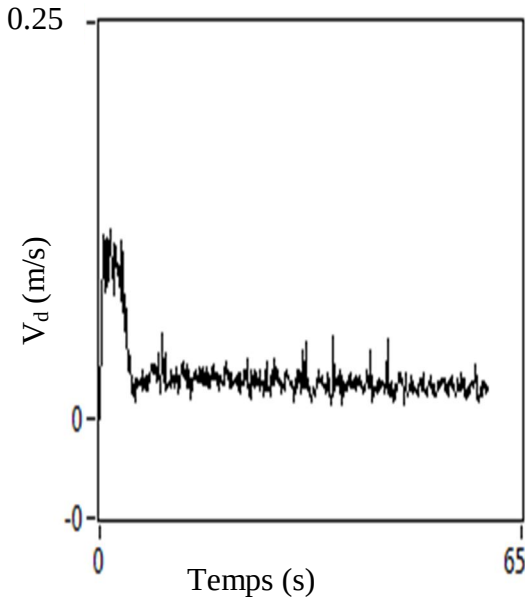


Figure.3.27.a : Trajectoire du robot



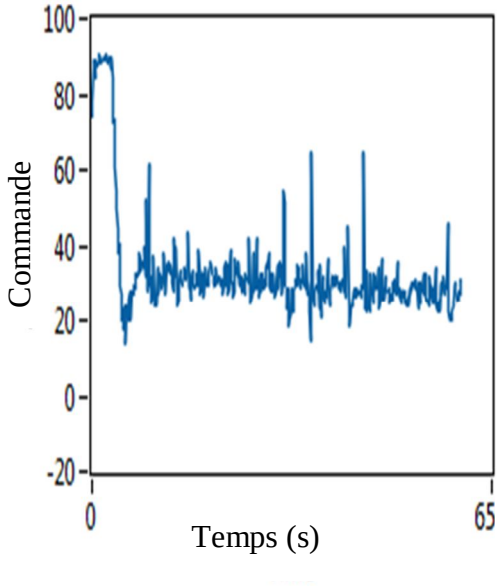
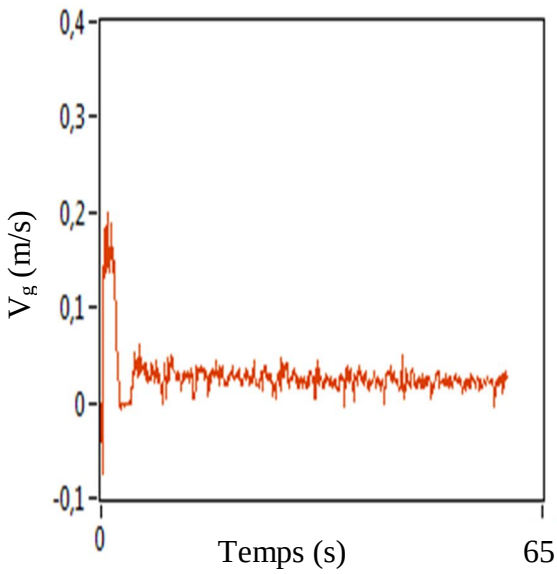


Figure.3.27.b : Vitesse de la roue droite **Figure.3.27.c:** Signal de commande de la roue droite



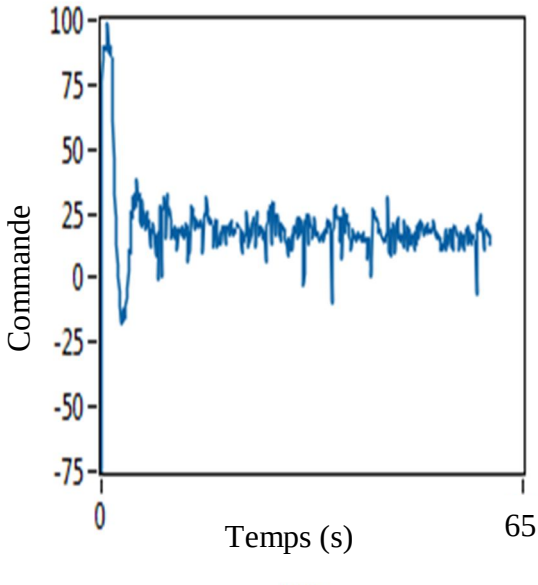


Figure.3.27.d : Vitesse de la roue gauche **Figure.3.27.e :** Signal de commande de la roue gauche

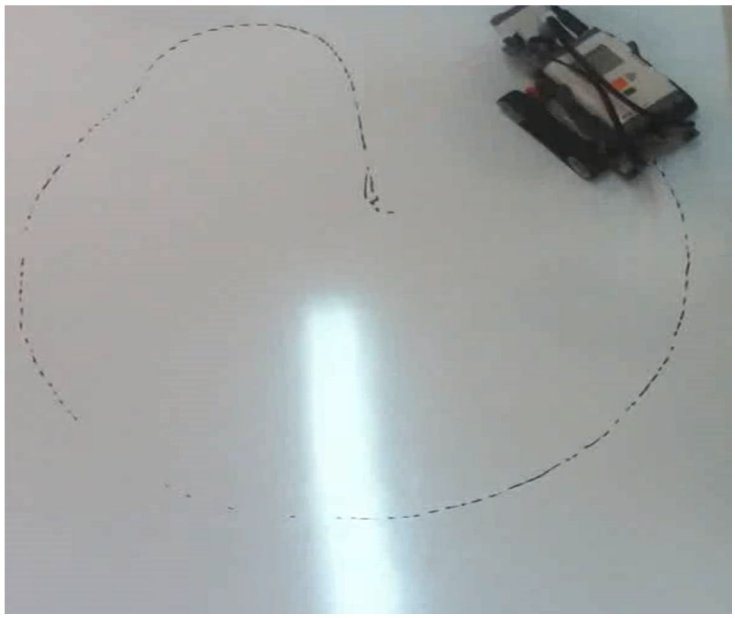


Figure.3.27.f : Le LEGO entraine de tracer la trajectoire

Les résultats du premier scénario sont présentés sur la figure 3.27 (a, b, c, d, e, f), où on peut remarquer que le robot LEGO suit la trajectoire de référence et la trace sur le sol. Les vitesses réelles (Figure 3.27 (b, d)) sont utilisées pour déterminer la position du robot LEGO en se basant sur (3.26), (3.27) et (3.28). Ce résultat montre la haute qualité du suivi.

3.7.3.2. Scénario 02 : suivi d'une trajectoire rectiligne et évitement d'obstacle

La trajectoire de référence est donnée par :

$$x_r(t) = at;\ y_r(t) = 0\ ;\ \omega_{0=}\ 0.15rad/s \qquad (3.29)$$

Les résultats du second scenario sont présentés dans la figure 3.28 (a, b, c, d, e, f, g):

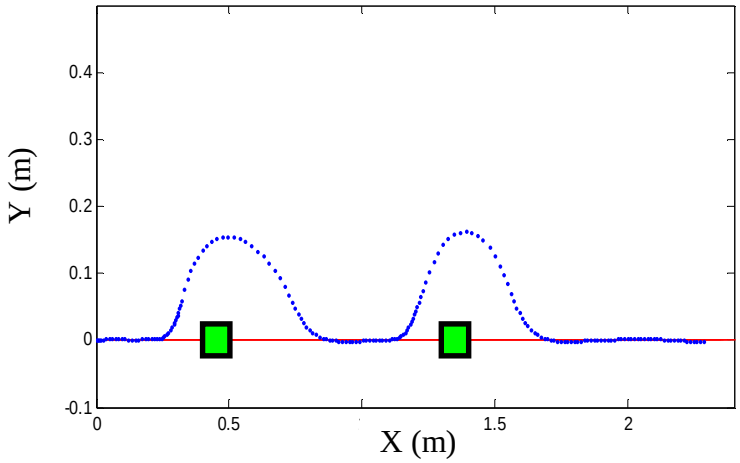
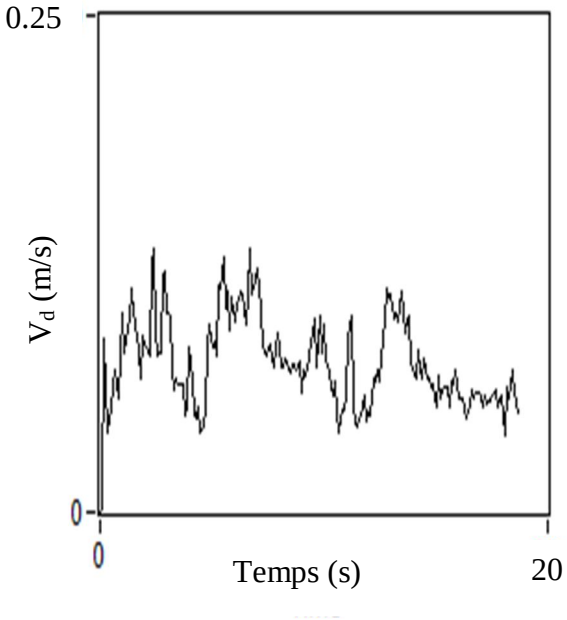


Figure.3.28.a : Trajectoire du robot



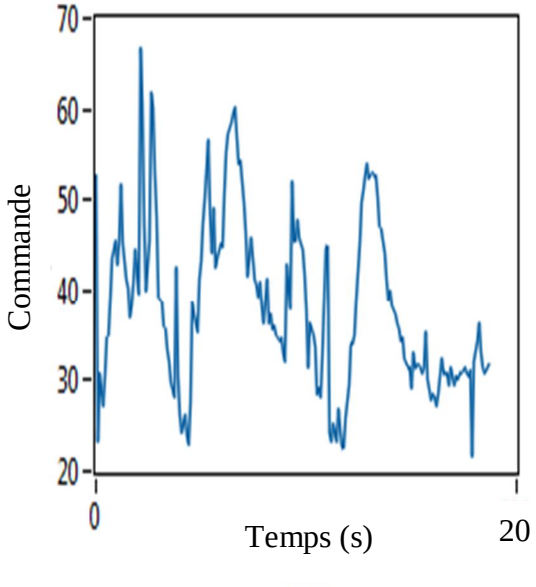
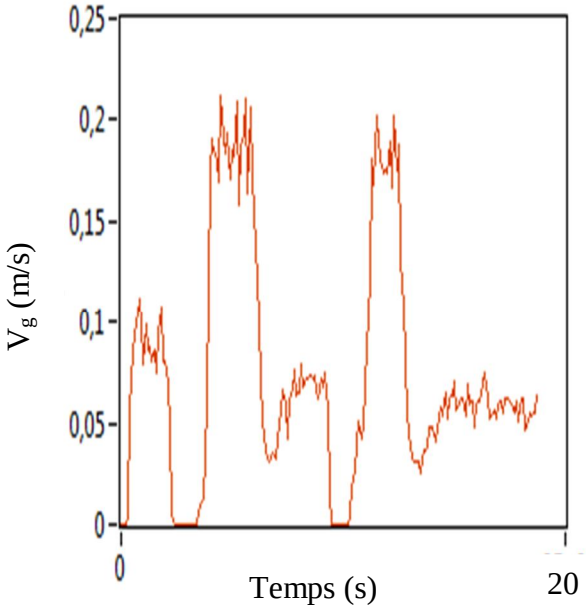


Figure.3.28.b : Vitesse de la roue droite **Figure.3.28.c :** Signal de commande de la roue droite



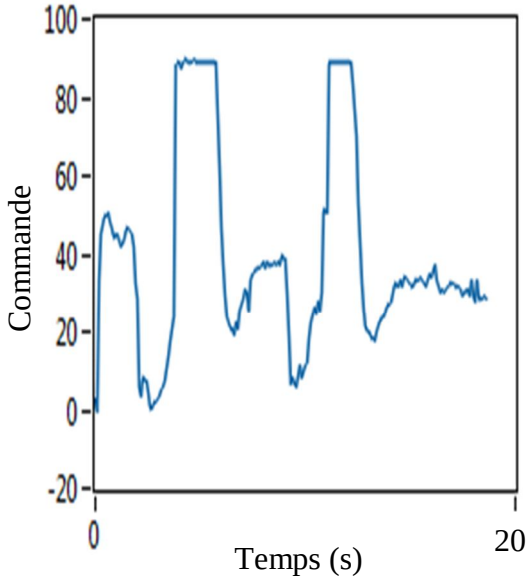


Figure.3.28.d : Vitesse de la roue gauche **Figure.3.28.e :** Signal de commande de la roue gauche

Figure.3.28.f : Le robot entrain de tracer la trajectoire **Figure.3.28.g :** Trajectoire tracée par le robot

On peut remarquer que le robot LEGO suit la trajectoire de référence et évite tous les obstacles ce qui prouve l'efficacité de cette approche.

3.8. Conclusion :

Dans ce chapitre, les algorithmes PSO, ACO et GSA ont été utilisés pour résoudre le problème d'optimisation qui apparait dans la commande prédictive linéaire et non-linéaire. Les résultats de simulations ont montrés l'efficacité de ces métaheuristiques. Cependant, l'algorithme PSO est très encourageant pour les applications en temps réel pour les systèmes qui ont une période d'échantillonnage de l'ordre de la milliseconde. Comme application expérimentale, le robot LEGO Mindstorms NXT a été commandé pour le suivi de trajectoire et l'évitement d'obstacles fixes par le biais d'un contrôleur prédictif qu'utilise l'algorithme d'optimisation PSO.

Chapitre 4

La Commande Prédictive Multi-Objectifs

4.1. Introduction :

Dans un grand nombre de problèmes du monde réel, il s'agit d'optimiser simultanément plusieurs objectifs pour un même problème. Ces objectifs sont souvent conflictuels. Ce qui exige d'abandonner le concept de " solution optimale" au profit des notions d'optimisation de vecteur.

Pour résoudre les défis des problèmes multi-objectifs, des algorithmes multi-objectifs qui sont en fait des extensions des algorithmes simples ont été proposés. Les plus connus sont les extensions des algorithmes génétiques au cas multi-objectif tels que MOGA : Multi-objectif Genetic Algorithm, NSGA : Non- dominated Sorting Genetic Algorithm, I et II ou l'extension de l'algorithme PSO tels que MOPSO : Multi-objectif PSO ou NSPSO : Non- dominated sorting PSO.

Dans ce chapitre, nous considérons l'optimisation multi-objectifs par l'algorithme PSO et GSA que nous allons utiliser pour résoudre le problème d'optimisation de la commande prédictive multi-objectifs.

4.2. L'optimisation multi-objectifs :

Le problème de l'optimisation multi-objectifs a la forme générale suivante :

$$min_{x\in\Omega}\{F(x)\} \tag{4.1}$$

Ω : est l'espace de variable de décision,

F : vecteur des fonctions objectifs

$$F: \Omega \to R^k, F(x) = [f_1(x), f_2(x), \dots, f_k(x)]^T$$

$$\text{Avec} \quad \begin{cases} g_i(x) \le 0 & i = 1,2,\dots,m \\ h_j(x) = 0 & j = 1,2,\dots,p \end{cases} \tag{4.2}$$

Et k est le nombre des fonctions objectifs, $f_i: R^n \to R$, x est le vecteur des variables de décision représenté par $x = [x_1, x_2, \dots, x_n]^T$

La solution de ce problème est l'ensemble particulier de valeurs $x^* = [x_1^*, x_2^*, \dots, x_n^*]^T$

A la fin du 19ème siècle, l'économiste *Vilfredo Pareto* formule le concept d'optimum de Pareto, qui constitue les origines de la recherche sur l'optimisation multi-objectifs :

4.2.1. Définition 1 : soit $x, y \in \Omega$, x domine y (dénoté par $x \prec y$) SSI :$f_i(x) \leq f_i(y)$ et $F(x) \neq F(y)$

$$(\text{c.à.d. } \exists r : 1 \leq r \leq m \; tq \;\; f_r(x) < f_r(y)) \tag{4.3}$$

4.2.2. Définition 2 : soit $x^* \in \Omega$, x^* est une solution optimale de Pareto s'il n'y a pas une autre solution $y \in \Omega$ tel que $y \prec x^*$

4.2.3. Définition 3 : l'ensemble des solutions Pareto optimales est défini par :

$$PS = \{x \in \Omega | x \; est \; une \; soltion \; optimal \; de \; pareto\}$$

Cet ensemble est appelé *l'ensemble de Pareto.* Et son image dans l'espace des critères ($PF = \{F(x) | x \in PS\}$) est appelé *la surface de Pareto* ou *le front de Pareto optimal.*

Ils existent plusieurs méthodes pour la résolution de problèmes d'optimisation multi-objectifs. Dans la littérature, on trouve deux classifications différentes des méthodes de résolution des problèmes multi-objectifs :

- Dans la première classe, on trouve les méthodes qui transforment les problèmes d'optimisation multi-objectifs en un problème mono objectif pour retourner une solution unique et les méthodes qui fournissent un ensemble de solution Pareto optimal.
- Dans la deuxième classe, on trouve les méthodes a priori, les méthodes interactives et les méthodes a posteriori.

Dans la suite de cette section, nous nous intéresserons aux métaheuristiques de la première classe.

4.3. Métaheuristiques pour l'optimisation multi-objectifs :

Les algorithmes évolutionnaires sont largement appliqués pour l'optimisation multi-objectifs. Ils utilisent une population constituée de plusieurs individus représentant des solutions potentielles du problème donné. La qualité des individus est mesurée par un mécanisme d'évaluation (fitness) et la population est évoluée à travers des opérateurs prédéfinis tel que la sélection, la mutation et le croisement. Ces opérateurs permettent d'éliminer certains individus et d'en créer de nouveaux. Dans cette classe on trouve : L'algorithme VEGA [45] (Vector Evaluated Genetic Algorithm), le MOGA[46] (Multi-

objectif Genetic Algorithm), le NPGA [47] (Niched-Pareto Genetic Algorithm), l'algorithme NSGAII [44] (Non Dominated Sorting Genetic Algorithm II).

Une autre classe des algorithmes n'utilise pas les opérateurs de mutation, de sélection, et de croisement. Dans cette classe on trouve les algorithmes stochastiques tel que : l'algorithme de colonie de fourmis (MOACO), l'algorithme de l'optimisation par essaim de particules [49] (MOPSO), l'algorithme de recherche gravitationnel [48] (MOGSA), etc.

Dans ce qui suit, nous décrivons l'algorithme NSGAII, l'algorithme MOPSO et l'algorithme MOGSA.

4.3.1. Programmation de l'algorithme génétique multi-objectifs NSGAII :

Le NSGAII (Non-dominated Sorting Genetic Algorithm II) est un algorithme génétique multi-objectifs développé par Deb et al. Cet algorithme étant parmi les algorithmes les plus utilisés non seulement dans le cadre de l'optimisation multi-objectifs mais aussi comme référence pour comparaison avec d'autres algorithmes [50, 51, 52], nous le décrivons avec détails dans ce qui suit bien qu'il ne soit pas utilisé dans cette thèse. Dans cet algorithme, Deb et al tentent de résoudre toutes les critiques faites sur NSGAI [53] : complexité de calcul, non-élitisme et le partage de paramètre (sharing).

Principe de l'algorithme NSGAII :

L'algorithme NSGA-II est un algorithme évolutif multi-objectifs qui utilise un type de mesure du surpeuplement autour de chaque individu afin d'assurer la diversité de la population. Il est considéré comme étant plus efficace que le NSGA I parce qu'il utilise une procédure de tri basée sur la non-dominance (plus rapide), utilise une approche élitiste qui permet de sauvegarder les meilleures solutions trouvées lors des générations précédentes et utilise aussi un opérateur de comparaison basé sur un calcul de la distance de 'surpeuplement'.

Pour une génération *t* donnée, une population de parents (P_t) de taille (N) est assemblée avec une population enfants (Q_t) de taille (N) générée de la population parents précédente à travers les opérateurs de croisement et de mutation pour former une population $R_t = P_t \cup Q_t$. Cet assemblage permet d'assurer l'élitisme.

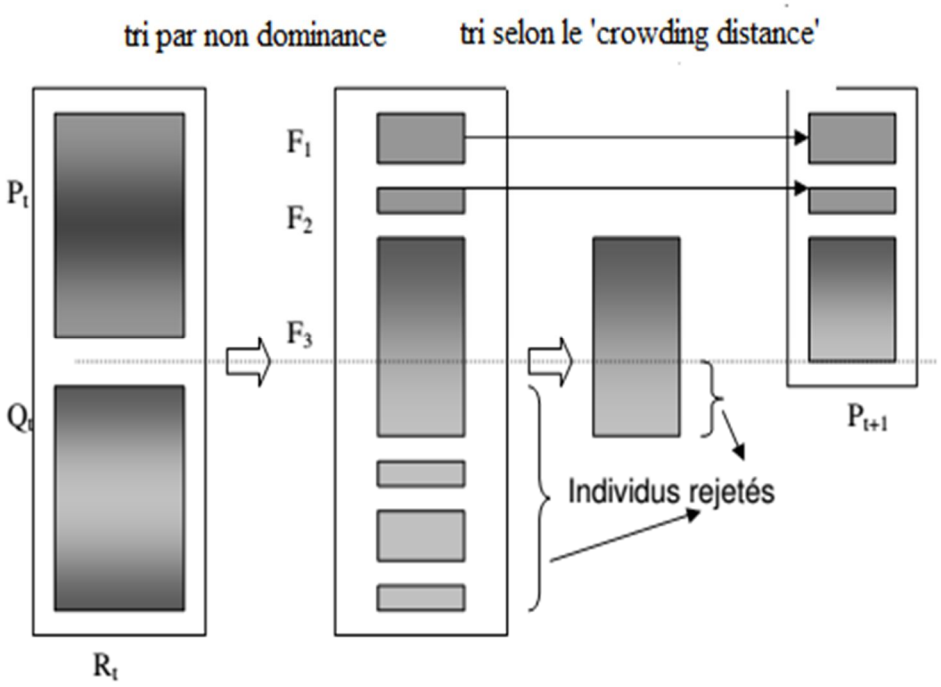


Figure.4.1 : Principe de l'algorithme NSGAII

La population R_t de taille ($2N$) est ensuite triée selon un critère de non-dominance pour identifier les différents fronts tel que F_1 représente les individus de rang 1, F_2 représente les individus de rang 2,etc. Donc les meilleurs individus vont se retrouver dans les premiers fronts. L'objectif suivant est de réduire le nombre d'individus de $2N$ dans la population R_t pour obtenir une population P_{t+1} de taille N. Si la taille de F_1 (ou les autres fronts) est inférieure à N, alors tous les individus de F_1 (ou les autres fronts) sont conservés (tant que ceux-ci ne dépassent pas N). Si l'on prend l'exemple de la figure 4.1, les fronts F_1et F_2 sont intégralement conservés mais la conservation du front F_3 va entraîner un dépassement de la taille N de la population P_{t+1}. Il faut alors procéder à une sélection des individus de F_3 à conserver.

Dans ce cas l'algorithme NSGA II utilise une procédure de calcul de ' surpeuplement' basée sur l'évaluation de la densité des individus autour de chaque solution pour préserver la diversité de la population. L'individu qui possède une faible valeur de 'surpeuplement' correspond à un individu bien entouré. On procède alors à un tri décroissant selon cette distance pour retenir des individus du front F_3 et éliminer ainsi les individus des zones les plus denses. De cette façon la population P_{t+1} est complétée. Ce mécanisme conserve les bornes extérieures du front de Pareto par la préservation des individus ayant des valeurs

extrêmes pour les critères. À la fin de cette phase, la population P_{t+1} est créée. Puis une nouvelle population Q_{t+1} est générée par reproduction à partir de P_{t+1}. On poursuit itérativement la procédure décrite ci-dessus jusqu'à la satisfaction de critère d'arrêt fixé par l'utilisateur.

La distance de 'surpeuplement' d'une solution (*i*) se calcul en fonction du périmètre formé par les points les plus proches de (*i*) sur chaque objectif. Le calcul de la distance de 'surpeuplement' nécessite tout d'abord le tri des solutions selon chaque objectif, dans un ordre ascendant. Après, pour chaque objectif, les individus possédant les valeurs limites (la plus petite et la plus grande valeur de la fonction objectif) se voient associés une distance infinie. Pour les autres solutions intermédiaires, la distance est égale à la différence normalisée des valeurs des fonctions objectifs de deux solutions adjacentes. La distance de 'surpeuplement' est la somme des distances correspondantes à chaque objectif.

4.3.2. L'algorithme PSO pour l'optimisation multi-objectifs

Après la réussite de l'algorithme PSO dans le domaine de l'optimisation mono-objectif, la méthode d'optimisation par essaim de particules a été étendue pour résoudre les problèmes d'optimisation multi-objectifs. Plusieurs approches ont été développées [54]. On trouve par exemple : "Multiple objective PSO with crowding distance and local search" (MOPSO-CDLS) [55], Multiple objective PSO with crowding distance and roulette wheel" (MOPSO – CDR)[56], "Two Sub-swarms Exchange PSO" (TSE-PSO)[57], « Non Dominated Sorting PSO »(NSPSO)[58].

Dans ce qui suit, on va présenter les deux algorithmes les plus connus pour l'algorithme PSO: le MOPSO et le NSPSO.

4.3.2.1. Principe du MOPSO :

Les principales caractéristiques de l'algorithme MOPSO (Multiple Objectives Particle Swarm Optimization), proposé par Coello [54] sont : l'emploi d'une archive externe pour la sauvegarde des solutions non-dominées, prise en compte des contraintes, la distance de surpeuplement, choix du guide cognitif.

L'analogie de l'optimisation par essaim de particules avec les algorithmes évolutionnaires met en évidence l'idée que l'utilisation d'une méthode de classement de Pareto pourrait être le moyen simple d'étendre l'approche pour traiter les problèmes d'optimisation multi-objectifs.

Deux mémoires sont utilisées, une pour stocker les *gbest* et l'autre pour les *pbest* de chaque particule de l'essaim. Un archivage est aussi utilisé pour stocker les particules non dominées de l'essaim. Un individu non - dominé de la population actuelle peut être inclus dans l'archive seulement s'il n'est pas dominé par aucune des autres solutions existantes déjà dans l'archive. D'un autre côté, s'ils existent des solutions dans l'archive qui sont dominées par le nouveau individu ils seront automatiquement exclues. C'est le concept de l'élitisme.
Cette archive est divisée dans des hyper-cubes (des cubes avec un nombre de dimensions égale aux nombres des fonctions objectives), chacun ayant une valeur de fitness liée à la densité des particules qu'il contient. Par exemple la fitness est le résultat de la division de 10 par le nombre des particules résidant dans le cube concerné. Si l'archive des particules non dominées dépasse sa limite spécifiée, les individus dans les régions les plus denses seront éliminés jusqu'à ce que l'archive arrive à la taille souhaitée.
Ensuite, la technique de roulette est appliquée sur les hyper-cubes, pour assurer la diversité dans la frontière du Pareto. Le *gbest* est choisi aléatoirement à partir de l'hyper-cube choisi par la technique de roulette.
Le *pbest* est mis à jour en basant sur la notion de dominance au sens de Pareto. Si la nouvelle solution domine *pbest* donc il prend la valeur de cette solution et dans le cas où les deux sont non dominés, la nouvelle valeur de *pbest* est choisie aléatoirement entre le *pbest* courant et le *pbest* du (t+1). Des améliorations ont été proposées pour le MOPSO incluant un opérateur de mutation, prise en compte les contraintes, etc….

4.3.2.2. Principe du NSPSO :

Dans le NSPSO, les principaux mécanismes de l'algorithme NSGAII [44] sont adaptés au PSO pour être utilisés dans l'optimisation multi-objectifs.
Le problème avec la forme de base PSO est que les comparaisons de dominance ne sont pas totalement utilisées dans l'opération de mise à jour de la meilleure solution trouvée par chaque particule. Pour surmonter ce problème et augmenter le niveau de partage d'information entre les particules d'un essaim, NSPSO [58] combine l'ensemble des *N* particules de la population avec les *Npbest* de ces particules pour former une population temporaire de *2N* particules.
Ensuite, des comparaisons, selon la règle de dominance, entre tous les 2*N* particules sont effectuées pour le tri de toutes les particules de la population en différents fronts comme dans

le NSGAII. Un rang est attribué pour chaque particule sur la base du front auquel elle appartient. Cette approche 'combinaison puis-comparaison', assure plus d'éléments non dominés. En plus du rang, un nouveau paramètre appelé la distance de surpeuplement est calculé pour chaque particule pour assurer une meilleure distribution des solutions non - dominées. La distance de surpeuplement est une mesure de la proximité d'une particule à ses voisins.

La meilleure *gbesti* global pour l'*ième* particule *Pi* est choisi aléatoirement de la partie supérieure du premier front. *N* particules vont être sélectionnées et considérées comme des *pbest*, selon la fonction coût et la distance de surpeuplement. Ces informations sont utilisées pour le calcul de la nouvelle vitesse et la nouvelle position de chaque particule dans l'itération suivante.

4.3.3. Optimisation multi-objectifs par Algorithme de recherche gravitationnel (MOGSA)

Dans l'algorithme de recherche gravitationnel multi-objectifs, MOGSA, [52] les objets s'attirent mutuellement par la force de gravité provoquant un mouvement global, de tous les objets, vers les objets ayant les masses les plus lourdes. Par conséquent, les masses coopèrent en utilisant une forme de communication directe, par la force gravitationnelle. Les masses lourdes, correspondent aux bonnes solutions, déplacent plus lentement que les plus légers. Dans le MOGSA, chaque masse a quatre caractéristiques : position, masse inertielle, masse gravitationnelle active et masse gravitationnelle passive. La position de la masse correspond à une solution du problème, et les masses de gravité et d'inertie sont déterminées en utilisant une fonction de forme physique. L'algorithme est évolué en ajustant les masses de gravité et d'inertie. Finalement, les masses seront attirées par la masse la plus lourde qui présente une solution optimale dans l'espace de recherche.

Plus spécifiquement, s'il y a des *N* objets dans l'ensemble de solution avec des positions x_i, *i=1…N.*À l'itération *t*, la masse de chaque objet est donnée par :

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)} \tag{4.4}$$

avec

$$m_i(t) = \frac{fitness_i(t) - wst(t)}{bst(t) - wst(t)} \tag{4.5}$$

$fitness_i(t)$ est la valeur de la fonction coût de l'objet *i* tandis que le *bst(t)* et le *wst(t)* sont, respectivement, la meilleure et la pire valeur de la fonction coût de tous les objets. Soit *Kbest* l'ensemble des *k* objets possédants les meilleures valeurs de la fonction coût, alors la force totale agissant sur un objet donné provient des objets plus lourds et est donnée par [42]

$$F_i(t) = \sum_{j \in Kbest} rand_j \, G(t) \frac{M_j(t) M_i(t)}{D_{ij}(t) + \varepsilon} (x_j(t) - x_i(t)) \tag{4.6}$$

Avec $rand_j \in [0,1]$ un nombre aléatoire, *G(t)* la constante gravitationnelle, $D_{ij}(t)$ la distance euclidienne entre les objets *i* et *j*. La constante gravitationnelle est donnée par:

$$G(t) = G_0 e^{-\alpha t / tmax} \tag{4.7}$$

G_0 est la valeur initiale, α est une constante, *tmax* est le nombre maximum d'itérations.
L'accélération de l'objet *i* à l'itération *t* est donnée par:

$$a_i = \frac{F_i(t)}{M_i(t)} \tag{4.8}$$

La position $x_i(t)$ et la vitesse $v_i(t)$ de l'objet à l'itération *t* sont données par:

$$v_i(t) = rand_i v_i(t) + a_i(t) \tag{4.9}$$

$$x_i(t) = x_i(t) + v_i(t+1) \tag{4.10}$$

Le MOGSA présenté dans [48] est un algorithme d'optimisation multi-objectifs qui classe la population en différents fronts de Pareto, le classement des individus est effectué en utilisant le front de Pareto et le concept de distance de surpeuplement de la NSGA-II. Afin de calculer le coût multi-objectifs, une polarisation linéaire *br* est appliquée au $r^{ième}$ élément utilisant l'expression: *br =1/r*, obtenant des valeurs de 1 jusqu'à 1 / N. Ainsi, une population classée selon la fonction coût est aussi obtenue.

4.4. La commande prédictive multi-objectifs

On considère le problème de régulation à l'origine du système non-linéaire discret suivant :

$$x(t+1) = f\big(x(t), u(t)\big) \tag{4.11}$$

Avec les contraintes :

$x(k) \in \mathbb{X} \subseteq R^n$ est le vecteur d'état, $u(t) \in U \subseteq R^m$ est le vecteur d'entré

$\mathbb{X}$, *U* sont des ensembles compacts et convexes avec :

$$0 \in int(U), 0 \in int(\mathbb{X}) \text{ et } f(0,0) = 0, \tag{4.12}$$

Le problème résolu par la commande prédictive multi-objectifs est un problème de régulation vers l'origine par la résolution du problème d'optimisation multi-objectifs suivant :

$$\min_{u} \quad J(x,U) \tag{4.13}$$
$$x(k+1) = f\big(x(k),u(k)\big)$$
$$x(k) \in \mathbb{X}, k = 1,\dots,N$$
$$u(k) \in U, k = 0,\dots,N-1$$
$$x(k+N) \in \Omega$$

Avec

$$J(U,x) = [J_0(U,x), J_1(U,x),\dots,J_l(U,x)]^T : R^S \times R^n \to R^{l+1} \tag{4.14}$$

est un vecteur de fonction, $l \geq 1$,

$$s = N * m, U[u'_0,\dots,u'_{N-1}] \tag{4.15}$$

Avec

$$J_i(U,x) = F_i\big(x(k+N)\big) + \sum_{k=j}^{k+N-1} L_i(x(j),u(j)) \;\; avec \;\; i = 1,\dots,l \tag{4.16}$$

$F_i\big(x(k+N)\big)$ est le coût sur l'état finale. Aussi, l'état final est soumis sous la contrainte d'être à l'intérieur de la région finale:

$$x(k+N) \in X_f \subset X$$

Le coût F_i et la région finale sont introduits pour garantir la stabilité de la commande prédictive.

Dans ce travail, nous proposons de résoudre le problème d'optimisation multi-objectifs par les métaheuristiques basées sur les algorithmes PSO et GSA, en l'occurrence MOPSO, NSPSO et MOGSA.

4.5. Résultats de simulation :

4.5.1. Exemple 1 :

Le premier exemple concerne un système linéaire multi variables discret instable. Le système est décrit sous la forme d'espace d'état suivant [59]:

$$x(k+1) = \begin{bmatrix} 1.3433 & 0.1282 & 0 \\ -0.1282 & 1.2151 & 0 \\ 0.1282 & 0.0063 & 1.2214 \end{bmatrix} x(k) + \begin{bmatrix} 0.1164 & 0.0059 \\ -0.0059 & 0.1105 \\ 0.1166 & -0.1105 \end{bmatrix} u(k) \tag{4.17}$$

Avec

$$-20 < u(k) < 20 \tag{4.18}$$

Le problème consiste à minimiser trois fonctions objectifs ($i=3$) (pour une régulation vers l'origine):

$$\min_{u}\left\{J_i(U,x(t)) = x^T_{t+\frac{N}{t}} P_i x_{t+\frac{N}{t}} + \sum_{k=0}^{N-1} x^T_{t+\frac{k}{t}} Q_i x_{t+\frac{k}{t}} + u^T_{t+k} R_i u_{t+k}\right\} \quad (4.19)$$

Pour assurer la stabilité, la matrice de coût final P_i est calculée à partir de l'équation algébrique de *Riccati* avec l'hypothèse que les contraintes ne sont pas actives pour $k \geq N$.

$$Q1 = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}; Q2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}; Q3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad (4.20)$$

$$R1 = R2 = R3 = I2;\ x(0) = [2\ 2\ -1]' \quad (4.21)$$

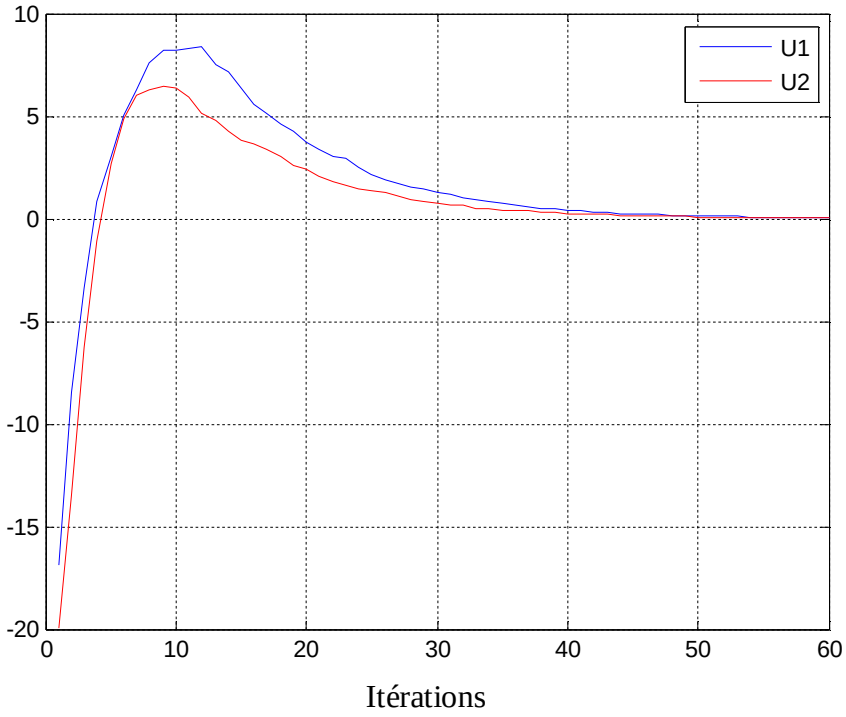


Figure.4.2 : Signaux de commande

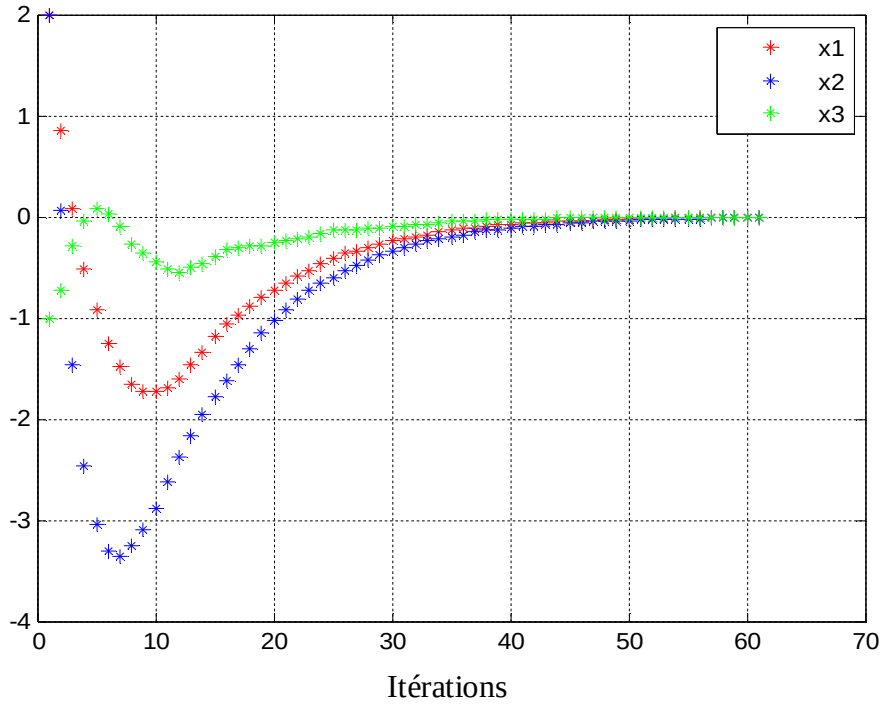


Figure.4.3 : Etats du système

Le signal de commande et les états sont tout à fait semblables pour les deux algorithmes (MOPSO-MPC, NSPSO-MPC). De plus, ces solutions sont similaires avec celles obtenus dans [59] utilisant une approche basée sur l'horizon glissant. Le système est stabilisé après quarante (40) itérations tout en satisfaisant les contraintes. Les résultats montrent que les algorithmes étudiés convergent vers une solution proche de l'optimal après un temps de calcul suffisant. Cependant, l'algorithme MOPSO-MPC converge plus rapidement. En outre, le MOPSO-MPC est simple à coder et converge rapidement par rapport aux autres algorithmes destinés pour l'optimisation multi-objectifs.

	MOPSO-MPC	NSPSO-MPC
Temps de calcul	≤30ms	≤115ms

Tableau4.1 : Temps de calcul pour les deux algorithmes

4.5.2. Exemple 2 :

Dans ce deuxième exemple, on considère quatre robots mobiles, deux robots réels et deux autres virtuels, les réels poursuivants les virtuels. On suppose qu'il n'y a pas de glissement (roulement pur).

Le modèle cinématique des robots réels est donné par :

$$\dot{x}_i(t) = \frac{v_{di}(t)+v_{gi}(t)}{2} cos\theta_i(t) \tag{4.22}$$

$$\dot{y}_i(t) = \frac{v_{di}(t)+v_{gi}(t)}{2} sin\theta_i(t) \tag{4.23}$$

$$\dot{\theta}_i(t) = \frac{v_{di}(t)-v_{gi}(t)}{b} \tag{4.24}$$

Où *i*=1, 2, $v_{di} \in R$ et $v_{gi} \in R$ sont les vitesses linéaire de la roue droite et la roue gauche, respectivement, du robot *i*. $b \in R$ est la distance entre les centres des roues. θ_i est l'angle d'orientation du robot *i*, ω_i est la vitesse angulaire du robot *i*.

Le problème est de trouver la loi de commande définie par $v_{di}(t)$, $v_{gi}(t)$ (*i=1, 2*) permettant au robots réels:

- De suivre les trajectoires de référence (robots virtuels) définis par : $[x_{ri}(t) y_{ri}(t)], i = 1,2$
- D'éviter les obstacles fixes sur les trajectoires
- D'éviter la collision entre les deux robots

Les trajectoires des robots virtuels sont définies par :$[x_{r1}(t) y_{r1}(t)]$, $[x_{r2}(t) y_{r2}(t)]$ respectivement.

$$x_{r1}(t) = \cos \omega_0 t,\ y_{r1}(t) = \sin(2 * \omega_0 t) \tag{4.25}$$

$$x_{r2}(t) = \cos(\omega_0 t + \varphi)\ ;\ y_{r2}(t) = \sin(2 * \omega_0 t + \varphi) \tag{4.26}$$

ω_0=0.02 rad/s est la pulsation du signal.

Les contraintes sur les signaux de commande sont :

$$-0.7(m/s) \leq v_{di} \leq 0.7\ (m/s) \tag{4.27}$$

$$-0.7(m/s) \leq v_{gi} \leq 0.7(\ m/s) \tag{4.28}$$

La période d'échantillonnage est de 100ms.

Ce problème est formulé comme un problème de commande prédictive multi-objectifs avec contraintes, résolu parle MOPSO et le MOGSA. L'évitement d'obstacle est assuré par l'ajout d'une contrainte au problème de la MOMPC : si la distance entre l'obstacle et le robot est

inférieure à une valeur donnée, alors une pénalité est ajoutée à la fonction coût. La même idée est utilisée pour éviter la collision entre les robots.

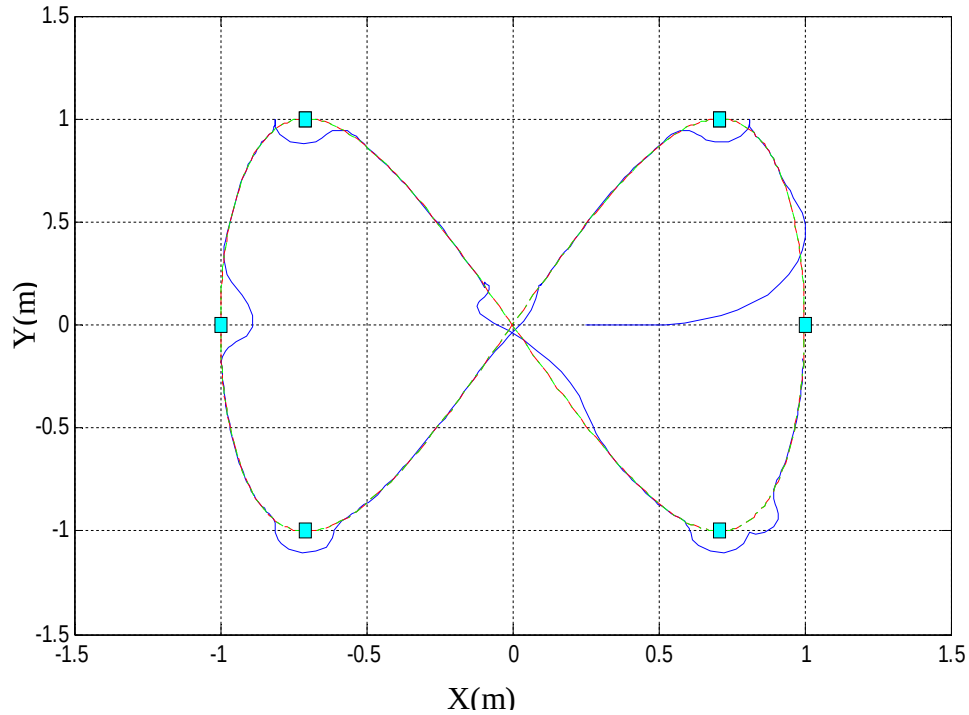


Figure.4.4 : Trajectoire du robot1

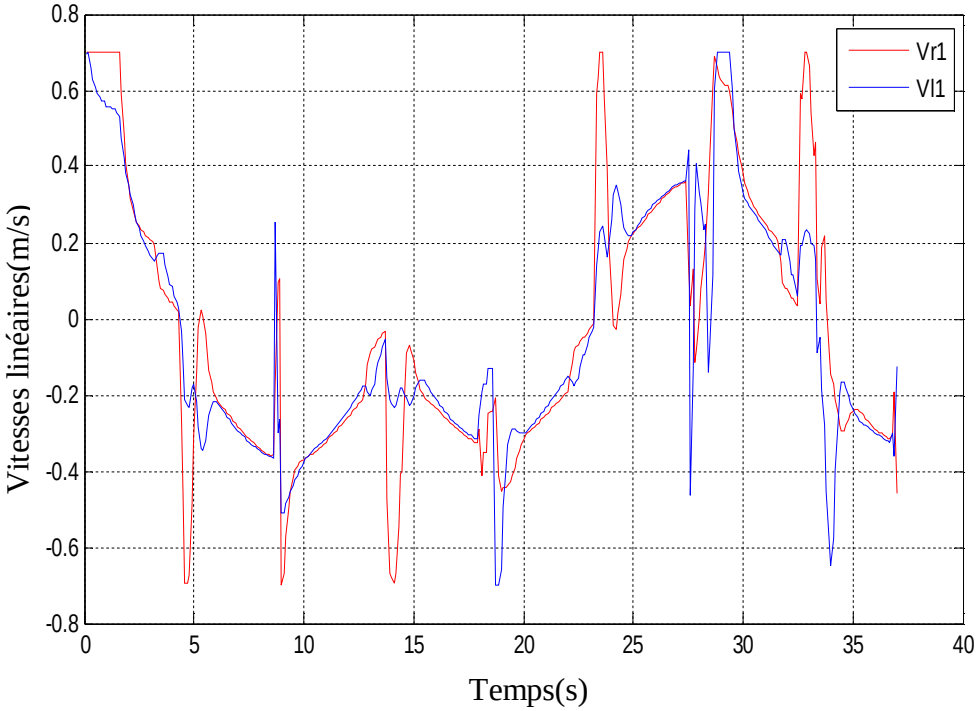


Figure.4.5 : Signaux de commande pour le robot1

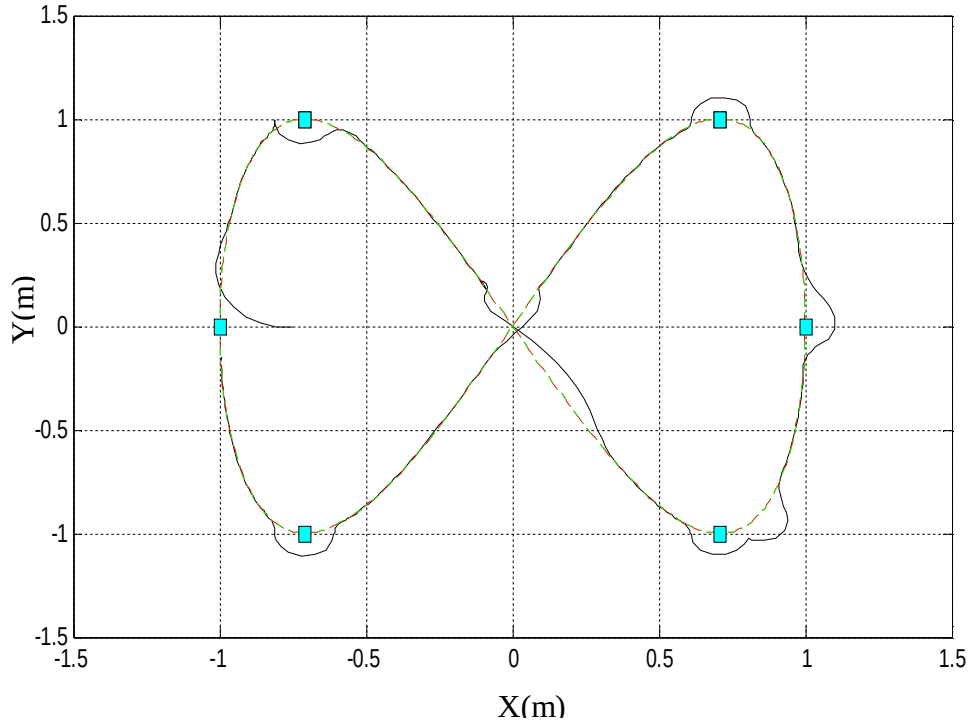


Figure.4.6: Trajectoire du robot2

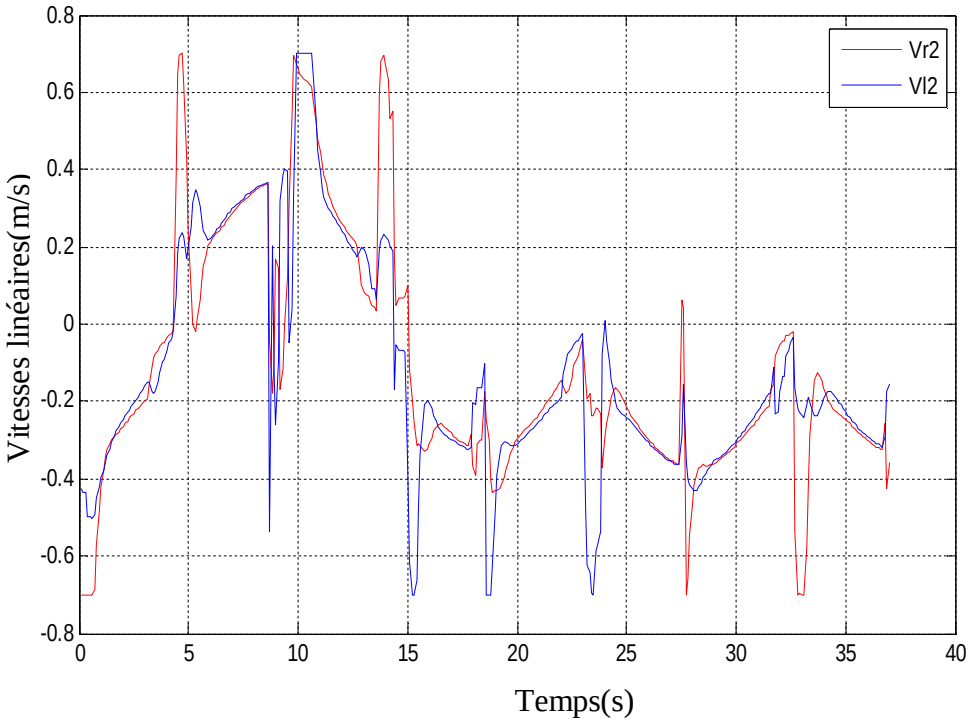


Figure.4.7 : Signaux de commande pour le robot2

Les deux algorithmes sont exécutés jusqu'à ce qu'une réponse satisfaisante soit obtenue. Le point de collision des robots est *m(0,0)*. Les robots commencent à partir de leurs positions de

départ (0,25, 0), (-0,75, 0), chacun suit sa propre trajectoire de référence. Chaque robot évite le premier obstacle fixe et poursuit son parcours au point de collision. Il est observé que le second robot continu son suivi et le premier l'évite en diminuant sa vitesse afin de maintenir une bonne distance de sécurité. Les robots poursuivent leur suivi et évitent les obstacles fixes rencontrés. Les temps de calcul sont donnés dans le tableau 4.2, où le MOPSO-MPC est meilleur que MOGSA-MPC qui produit un suivi et un évitement similaire mais avec un temps de calcul plus long. Les contraintes sur les signaux de commande sont toujours satisfaites comme indiqué sur la figure 4.5 et la figure 4.7. Les simulations sont effectuées sur Intel® I3, 2120 CPU 3,30 GHz avec 4Go de RAM.

	MOPSO-NMPC	MOGSA-NMPC
Temps de calcul	$\leq$ 9ms	$\leq$ 190ms

Tableau 4.2: Temps de calcul pour l'exemple 2

4.6. Conclusion :

Dans ce chapitre, nous avons étudié l'applicabilité des métaheuristiques multi-objectifs pour la solution en ligne de la commande prédictive multi-objectifs linéaire et non-linéaire. Pour ceci, nous avons présenté quatre heuristiques différentes : le NSGAII, le MOPSO, le NSPSO et le MOGSA. Deux exemples ont été considérés : commande d'un système linéaire multi variable et commande de deux systèmes non-linéaire multi variable (robots mobiles).

Les résultats montrent que les métaheuristiques étudiées convergent vers une solution proche de l'optimal si on leur donne un temps de calcul suffisant. Cependant, l'algorithme MOPSO converge plus rapidement et est encourageant pour la commande des systèmes en temps réel par le biais de la commande prédictive. En outre, le MOPSO est simple à coder et peut être une alternative à NSGAII et d'autres méthodes qui sont généralement plus difficiles à mettre en œuvre.

Conclusion Générale

Conclusion générale

Les objectifs du travail développé dans cette thèse étaient une étude de la robustification de la commande prédictive à base de modèle et la réduction du temps de calcul de la solution du problème d'optimisation de la commande. La robustification consiste à réduire les effets de variations de paramètres sur la qualité de la solution. La réduction du temps de calcul permet d'étendre le champ d'application aux systèmes dits rapides qu'on trouve en particulier en robotique, mécanique, etc...

Dans la première partie de la thèse, un contrôleur prédictif robuste basé sur la théorie des intervalles flous est proposé. Les intervalles flous sont utilisés pour décrire les incertitudes structurées. En se basant sur les alpha-coupes et la sortie mesurée, le modèle de prédiction est mis à jour à chaque période d'échantillonnage puis employé par la MPC pour définir le signal de commande. Cette approche est testée sur deux exemples : la commande de la vitesse angulaire d'un moteur à courant continu et la commande de la concentration dans un réacteur chimique. La méthode a été comparée avec la commande prédictive non robuste (simple) qui ne tient pas en compte les variations sur les paramètres. Les résultats obtenus ont montrés l'insensibilité des systèmes commandés aux variations des paramètres démontrant la robustesse de cette approche par rapport à la commande prédictive simple. L'inconvénient de cette méthode c'est qu'elle est coûteuse en termes de temps de calcul.

Dans la deuxième partie, l'applicabilité des métaheuristiques pour déterminer la solution optimale en ligne de la commande prédictive linéaire et non-linéaire a été étudiée. Une comparaison entre trois métaheuristiques : algorithme de colonie de fourmis, ant colony optimisation, ACO, optimisation par essaim de particules, Particle swarm Optimisation, PSO, et l'algorithme de recherche gravitationnel, Gravitational Search Algorithm, GSA, a été effectuée. Ces métaheuristiques ont été appliquées à la solution de la MPC pour commander des systèmes : linéaire mono variable, linéaire multi variable, non-linéaire mono variable et non-linéaire multi variable. Les meilleurs résultats en termes de temps de calcul ont été obtenus avec l'algorithme PSO. Ceci est très encourageant pour les applications en temps réel pour les systèmes qui ont une période d'échantillonnage de l'ordre de la milliseconde. L'algorithme PSO a été appliqué par une étude expérimentale à la commande d'un robot mobile à deux roues de type LEGO Mindstorms NXT, qui est un système non-linéaire multi variable, pour le suivi de trajectoire et l'évitement d'obstacles.

Par ailleurs, dans un grand nombre de problèmes du monde réel, il s'agit d'optimiser simultanément plusieurs objectifs d'un même problème souvent conflictuels et leur résolution dans un temps raisonnable devient nécessaire. C'est le cas de la commande prédictive multi-objectifs. Une étude en simulation été effectuée sur l'applicabilité des métaheuristiques multi-objectifs pour la solution en ligne de la MPC multi-objectifs linéaire et non-linéaire. Pour ceci, deux exemples ont été considérés, la commande d'un système linéaire multi variable et la commande de deux systèmes multi variable simultanément (la commande de deux robots mobiles pour le suivi de trajectoire et l'évitement d'obstacles). Les résultats montrent que les métaheuristiques étudiées convergent vers une solution proche de l'optimal si on leur donne un temps de calcul suffisant. Cependant, l'algorithme MOPSO converge plus rapidement et est encourageant pour la commande prédictive en temps réel. En outre, le MOPSO est simple à coder et peut être une alternative à NSGAII et d'autres méthodes qui sont généralement plus difficiles à mettre en œuvre.

Comme perspective, la méthode de robustification développée pour les systèmes linéaire sera étendue vers les systèmes non-linéaires.

Références

[1] D.Q. Mayne, J.B. Rawlings, C.V. Rao and P.O.M. Scokaert,"Constrained model predictive control: Stability and optimality", Automatica, Vol. , N° 36, pp. 789-814, 2000.

[2] E.F. Camacho and C. Bordons, "Model Predective Control", Springer-Verlag Berlin hedelberg New York , 1999

[3] J.A. Rossiter, "Model Based Predictive Control A Practical Approach", CRC PRESS London , 2004.

[4] M. Maciejowski, "Predictive control with constraints", Prentice Hall, 2001

[5] A. Bemporad, M. Morari, V. Dua and E.N. Pistokopoulos, "The explicit linear quadractic regulator for constrained systems," Automatica, 38(1), 3-20, 2002.

[6] A. Bemporad, A. Oliveri, T. Poggi, and M. Storace, "Ultra-Fast Stabilizing Model Predictive Control via Canonical Piecewise Affine Approximations," IEEE Transactions on Automatic Control, vol. 56, N° 12, pp. 2883-2897, 2011.

[7] M. Kogel and R.Findeisen, "Fast predictive control of linear systems combining Nesterov's gradient method and the method of multipliers", 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC), Orlando, FL, USA, December 12-15, 2011

[8] R. C. Eberhart and Shi, Y., "Particle swarm optimization: developments, applications and resources," Proc. Congress on Evolutionary Computation Seoul, Korea, 2001.

[9] M. M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization artificial ants as a computational intelligence technique", IEEE Computational intelligence Magazine, 1989.

[10] K. Socha , M. Dorigo, "Ant colony optimization for continuous domains," European Journal of Operational Research, no. 185, 2008, pp. 1155–1173.

[11] L. Wang "Model Predictive Control System Design and Implementation Using MATLAB", Springer, 2009

[12] C. V. Rao, S. J. Wright, J. B. Rawlings, "Application of Interior- Point Methods to Model Predictive Control", journal of optimization theory and applications: vol.99,no.3. pp.723-757, december1998

[13] A. Bemporad and C. Filippi, "Suboptimal explicit receding horizon control via approximate multiparametric quadratic programming", J. Optim. Theory Applicat., vol. 117, no. 1, pp. 9–38, Apr. 2003.

[14] M. Canale, L. Fagiano, and M. Milanese, “Set membership approximation theory for fast implementation of model predictive control”, Automatica, vol. 45, no. 1, pp. 45–54, 2009.

[15] M. Diehl, H. Bock, D. Leineweber, J. Schlooder, “Efficient direct multiple shooting in nonlinear model predictive control”, in: F. Keil, W. Mackens, H. Vos, J. Werther (Eds.), Scientific Computing in Chemical Engineering II, Vol. 2, Springer, Berlin, 1999.

[16] A. Casavola, Famularo D. and G. Franzé G. “Predictive control of constrained nonlinear systems via LPV linear embeddings”, Int. J. Robust and Nonlinear Control, vol.13,pp. 281-294, 2003.

[17] G. A. Franzé, “Nonlinear Sum-of-Squares Model predictive Control Approach”. IEEE Transaction on Automatic Control”, vol. 55, pp.1466-1471, 2010.

[18] C. C. Chen and L. Shaw, “On receding horizon control”, Automatica, Vol 16,pp.349-352,1982.

[19] D.Q. Mayne, and H. Michilska, “Receding horizon control of non-linear systems”, IEEE Tansactions on Automatic Control, Vol. 5, N° 35, pp. 814-824, 1990

[20] S. Keerthi and E. G. Gilbert, “Optimal infinite horizon feedback laws for a general class of constrained discrete time systems: Stability and moving-horizon approximations”, Journal of Optimization Theory and Applications, Vol. 57, pp. 265–293, 1988.

[21] D.Q. Mayne, J. B. Rawlings, C. V. Rao, "Constrained model predictive control: Stability and optimality", Automatica , N°36, pp.789-814,2000

[22] H. N. Nounou and K. M. Passino, “ Fuzzy model predictive control: techniques, stabilities issues and examples,” Proc. IEEE Int. Symp. Intelligent control, intelligent systems and Semiotics, pp. 423-428, 1999.

[23] D. Dubois, H. Prade, “Random sets and fuzzy interval analysis,” Fuzzy Sets and Systems, 42, pp.87–101, 1991.

[24] R.E. Moore, R. B. Kearfott, M.J. Cloud, “Introduction to Interval Analysis”, Society for Industrial and Applied Mathematics, 2009.

[25] R. Boukezzoula, L. Foulloy & S. Galichet, “Inverse controller design for fuzzy interval Systems”, IEEE Transaction on Fuzzy Systems,14, pp.111-123, 2006.

[26] M. A. BISSERIER, thèse, "Une approche paramétrique de la régression linéaire floue - Formalisation par intervalles", UNIVERSITE DE SAVOIE, 2011.

[27] V. Ghaffari, S. Vahid Naghavi, , A.A. Safavi, "Robust model predictive control of a class of uncertain nonlinear systems with application to typical CSTR problems", Journal of Process Control 23, 493– 499, 2013.

[28] A. Bemporad, & M. Morari, "Robust model predictive control: A survey". In A. Garulli, A. Tesi, & A. Vicino (Eds.), Robustness in identification and control (Vol. 245) (pp. 207–226), Lecture notes in control and information sciences. Berlin: Springer, 1999

[29] J. C. Allwright and G. C. Papavasiliou. "On linear programming and robust model-predictive control using impulse-responses". Syst. Control Let, 18, 159-164,1992

[30] M. Canale, & M. Milanese, "Robust design of predictive controllers in presence of unmodeled dynamics". European Journal of Control, 9(5), 499–506, 2003.

[31] A, Casavola, D. Famularo,& G. Franzé, "Robust constrained predictive control of uncertain norm-bounded linear systems". Automatica, 40(11), 1865–1876, 2004.

[32] C. Løvaas, M. M. Seron, Graham C. Goodwin, "Robust output-feedback model predictive control for systems with unstructured uncertainty", Automatica 44, 1933–1943, 2008

[33] C. Ramos, M. Martínez, J. Sanchis, J.M. Herrero, " Robust and stable predictive control with bounded uncertainties", J. Math. Anal. Appl. 342 1003–1014, 2008

[34] D. Muñoz de la Peña, A. Bemporad, and C. Filippi, " Robust Explicit MPC Based on Approximate Multiparametric Convex Programming", IEEE Transactions On Automatic Control, Vol. 51, No. 8, 2006

[35] M. Kothare, V. Balakrishnan, & Morari, M, "Robust constrained model predictive control using linear matrix inequalities". Automatica, 32, 1361–1379, 1996.

[36] B. Kouvaritakis, J. A. Rossiter, & J. Schuurmans, " Efficient robust predictive control". IEEE Transactions on Automatic Control, 45,1545–1549, 2000.

[37] S. G. Cao, N. W. Rees, and G. Feng, "Analysis and design for a class of complex control systems – Part I: Fuzzy modeling and identification", Automatica, vol. 33, pp. 1017–1028, 1997.

[38] W. Yang, G. Feng, T. Zhang, "Robust Model Predictive Control for Discrete-Time Takagi-Sugeno Fuzzy Systems with Structured Uncertainties and Persistent Disturbances", IEEE Transactions on Fuzzy Systems, 2013.

[39] W. Langsona, I. Chryssochoosb, S.V. Rakovi,cb, D.Q. Mayne, "Robust model predictive control using tubes", Automatica, 40, 125 – 133, 2004.

[40] K. Chris WU, "Fuzzy interval control of mobile robots", Computers Elect. Engng ,Vol. 22, No. 3, pp. 211 -229, 1996.

[41] N. Nedjah, L. M. Mourelle, "Swarm Intelligent Systems", Springer-Verlag Berlin Heidelberg, 2006.

[42] E. Rashedi, H. Nezamabadi-pour , S. Saryazdi, "GSA: A Gravitational Search Algorithm", Information Sciences , no. 179, pp. 2232–2248, 2009.

[43] P. Thondel, T. Johansen, A. Bemporad, "An algorithm for multi-parametric quadratic programming and explicit MPC solutions," Automatica no. 39, pp. 489 – 497, 2003.

[44] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan , "A fast elitist non-dominated sorting genetic algorithm for multi-objectif optimization: NSGA-II", Parallel Problem Solving from Nature (PPSN VI), Berlin, pp. 849-858, 2000.

[45] J. D. Schaffer, "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms", In genetic Algorithm and their Applications: Proceedings of the First International Conference on Genetic Algorithm, Hillsdale, NJ, USA, p.93 -100, 1985.

[46] C. M. Fonseca, P. J. Fleming, "Genetic algorithms for multi-objectif optimization: formulation, discussion and generalization", Proceedings of the 5th International Conference on Genetic Algorithms, San Mateo, California, pp. 416-423, 1993.

[47] J. Horn, N. Nafpliotis, D. E. Goldberg, "A niched Pareto genetic algorithm for multi objective optimization", Proceedings of the 1st IEEE Conference on Evolutionary Computation, IEEE World Congress on Evolutionary Computation, vol. 1, pp. 82-87, Piscataway, NJ, IEEE Press, 1994.

[48] J. Cabestany, I. Rojas, and G. Joya (Eds.): "Applying a Multi objective Gravitational Search Algorithm (MO-GSA) to Discover Motifs", IWANN 2011, Part II, LNCS 6692, pp. 372–379, 2011. Springer-Verlag, Berlin Heidelberg, 2011

[49] N.A.A. Aziz, M.Y. Alias, A. W. Mohemmed, K.A. Aziz, "Particle Swarm Optimization for Constrained and Multiobjective Problems: A Brief Review", International Conference on Management and Artificial Intelligence, IPEDR vol.6, 2011.

[50] R. Furtuna, S. Curteanu, Florin Leon, "Multi-objectif optimization of a stacked neural network using an evolutionary hyper-heuristic", Elsevier B.V. Applied Soft Computing, 2011.

[51] H. Li; Q. Zhang, "Multiobjective Optimization Problems With Complicated Pareto Sets, MOEA/D and NSGA-II", IEEE Transactions on Evolutionary Computation, Vol. 13, Issue. 2, pp 284 – 302, 2009.

[52] Y. Wang, X. Lei, Y. Jiang, H. Wang, "Performance comparison of three multi-objectif optimization algorithms on calibration of hydrological model", Sixth International Conference on Natural Computation (ICNC), Vol. 6, pp. 2798 - 2803 , 2010 .

[53] K. Deb, A. Pratap, S. Agarwal, and T. Meyariva "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II", IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 6, NO. 2, APRIL, 2002.

[54] M.R. Sierra, , C.A. Coello Coello, "Multi-objectif particle swarm optimizers: asurvey of the state-of-the-art", International Journal of Computational Intelligence and Research, 2 (3), 287–308, 2006

[55] C.S. Tsou, S.C. Chang, and P. W. Lai, "Using crowding distance to improve multi-objectif PSO with local search", Swarm Intelligence, Focus on Ant and Particle Swarm Optimization, pp. 77-86, 2007.

[56] R. A. Santana, M. R. Pontes, and C.J.A. Bastos-Filho, "A multiple objective particle swarm optimization using crowding distance and roulette wheel", in International Conference on Intelligent Systems Design and Applications ISDA, Pisa, , pp. 237-242, 2009

[57] J. Zhao, L Lu, and H. Sun, "A modified two sub-swarms exchange particle swarm optimization", in International Conference on Intelligent Technology and Automation , , pp. 180-183,2010

[58] X. Li, "A non-dominated sorting particle swarm optimizer for multiobjective optimization", Proc. of Genetic and Evolutionary Computation, In Springer-Verlag Lecture Notes in Computer Science, 2723, pp. 37-48, 2003

[59] D. De Vito & R. Scattolini, "A receding horizon approach to the multiobjective control problem", In Proc. 46th IEEE conf. on decision and control. pp. (6029-6034), 2007.

[60] C. M. Huang, S. P. Yang, H. T. Yang, and Y. C. Huang, "Combined particle swarm optimization and heuristic fuzzy inference systems for a smart home one-step-ahead load forecasting." Journal of the Chinese Institute of Engineers, 37(1), pp. 44–53, 2014.

[61] H. C. Huang, S. S. D. Xu, and H. S. Hsu. “Hybrid Taguchi DNA Swarm Intelligence for Optimal Inverse Kinematics Redundancy Resolution of Six-DOF Humanoid Robot Arms.” Mathematical Problems in Engineering, pp.1-9, 2014.

Résumé

Dans ce travail, un contrôleur prédictif robuste basé sur la théorie des intervalles flous est introduit. Les paramètres variables du modèle du système sont représentés par des intervalles flous. En se basant sur les alpha-coupes et la sortie mesurée, à chaque période d'échantillonnage, un nouveau modèle de prédiction est construit et employé par la MPC pour définir le signal de commande. Cette méthode est cependant coûteuse en termes de temps de calcul. Par ailleurs, dans la deuxième partie de cette thèse, nous analysons l'application des métaheuristiques pour la détermination de la solution optimale en ligne de la commande prédictive. Pour cela, une comparaison entre trois métaheuristiques a été effectuée: algorithme de colonie de fourmis, optimisation par essaim de particules et l'algorithme de recherche gravitationnel. Les résultats montrent que l'algorithme d'essaim de particules converge plus rapidement. Celui-ci a été appliqué (par une étude expérimentale) pour la commande d'un robot mobile à deux roues. Après, une étude par simulation a été réalisée sur l'applicabilité des métaheuristiques multi objective pour la solution de la commande prédictive multi objective. Les résultats obtenus montrent que l'algorithme d'optimisation par essaim de particules multi objectif est encourageant pour les applications temps réel et peut être une alternative aux autres méthodes qui sont généralement plus difficiles à mettre en œuvre.

Mots clé : commande prédictive, intervalle flou, métaheuristiques, optimisation multi-objectifs.

ملخص

في هذا العمل، تم تطوير وحدة تحكم تنبؤيه قوية تستند إلى نظرية المجالات المبهمة وعيب هذه الطريقة هو أنها مكلفة للغاية من حيث الوقت. لتجاوز هذا الاشكال، درسنا إمكانية تطبيق طرق استكشاف لإيجاد الحل الامثل في الزمن الحقيقي للتحكم التنبئي. لهذا تم اجراء مقارنة بين ثلاث طرق استكشاف: خوارزمية مستعمرة النمل، خوارزمية سرب الجزيئات، خوارزمية البحث بالجاذبية. أظهرت النتائج أن خوارزمية سرب الجزيئات تقترب من الحل الامثل بشكل اسرع. بعد ذلك، تم تطبيق هذه الأخيرة للتحكم بروبوت متحرك ذو عجلتين. بعدها تم إجراء دراسة محاكاة لإمكانية تطبيق طرق الاستكشاف متعددة الأهداف لإيجاد الحل للتحكم التنبئي متعددة الأهداف. اثبتت النتائج أن خوارزمية سرب الجزيئات متعددة الأهداف يمكن اعتبارها بديلا للطرق الاخرى التي عادة ما تكون صعبة التنفيذ. كما أن النتائج جد مشجعة لاستخدام هذه الخوارزمية للتحكم في الأنظمة في الزمن الحقيقي.

كلمات مفتاحية: التحكم التنبئي، المجال المبهم، طرق الاستكشاف، التحسين متعدد الأهداف

Abstract

In this work, a robust predictive controller was developed based on fuzzy intervals theory. The drawback of this method is that it is time consuming. To raise this problem, the applicability of metaheuristics to determine the online predictive control optimal solution was studied. For this, a comparison between three metaheuristics was carried out: ant colony optimization and particle swarm optimization and gravitational search algorithm. Results show that particle swarm optimization algorithm converges faster. The latter was applied (by an experimental study) for the control of two wheel mobile robot. After that, a simulation study was carried out on the applicability of multi objective metaheuristics for solving multi objective predictive control. The results show that the multi objective particle swarm optimization algorithm is encouraging for real-time applications and can be an alternative to other methods which are generally more difficult to implement.

Key words: predictive control, fuzzy interval, metaheuristics, multi objective optimization.