UNIVERSITE DE BLIDA 1

Faculté de Technologie

Département d'Electronique

THESE DE DOCTORAT

en Automatique

Contribution to intelligent control

Par

Mohamed BENRABAH

Devant le jury composé de :

M. BOUNEKHLA	Professeur, U. de Blida 1	Président
D.E. KHODJA	Professeur, U. de M'Sila	Examinateur
A. GUESSOUM	Professeur, U. de Blida 1	Examinateur
R. BRADAI	MC A, U. de Blida 1	Examinateur
N. CHEGAGA	MC A, U. de Blida 1	Examinatrice
K. KARA	Professeur, U. de Blida 1	Rapporteur

U. Blida 1, 13 Juillet 2021

Remerciements

Tous mes remerciements vont à Dieu qui m'a guidé et m'a donné la patience et le courage pour finaliser ce projet.

Je tiens ensuite à remercier mes parents pour leur amour, leur soutien et leurs encouragements qui m'ont toujours donné confiance et motivation pour aller de l'avant, ainsi qu'à mes frères et mes sœurs, sans oublier de remercier mes amis pour leur présence et leur amitié sincère.

Je tien a exprimer ma sincère gratitude à mon directeur de thèse professeur KARA Kamel, directeur du laboratoire des systèmes électriques et télécommande, pour m'avoir au sein du laboratoire. Je le remercie pour son aide, son soutien, ses conseils précieuses tout au long de ce projet de recherche.

Je souhaite également remercie l'ensemble du membre du juré pour leurs instructions et conseils riches qu'ils mon partagés.

Enfin, permettez-moi d'espérer que ce travail soit une étape vers un long cheminement plein de sens et de réussite dans la recherche scientifique.

ملخص

الغرض الرئيسي من هذا العمل هو تطوير خوارزميات تحكم فعالة، بسيطة وقوية بحيث يمكن تطبيقها على عدد كبير من الأنظمة غير الخطية. تكمن الفكرة في تحسين أداء وحدات التحكم المعروفة والشائعة، وهي وحدة التحكم PID والتحكم التنبئي، وذلك باستخدام أدوات الذكاء الاصطناعي، مثل الشبكات العصبية، المنطق الضبابي، وطرق التحسين الاستكشافية الفوقية (meta heuristic optimization). سمحت دراسة طرق التحسين الاستكشافية الفوقية بتحديد طريقة التحسين المناسبة التي يمكن استخدامها في تطبيقات التحكم في الوقت الفعلي مع تقديم أداء جيد. في الواقع، سمحت هذه الدراسة أيضًا باقتراح تحسين لخوارزمية التحسين القائمة على التعلم (teaching learning based optimization. علاوة على ذالك، سمح العمل البحثي الذي تم إجراؤه باقتراح العديد من خوارزميات التحكم، وهي وحدة التحكم PID التكيفية ذات للشبكة العصبية، وحدة التحكم PID التكيفية ذات الشبكة العصبية من نوع سلسلة فورييه، والتحكم التنبئي ذو النموذج العصبي باستخدام طريقة التحسين القائمة على التعلم.

من أجل تحسين سرعة تقارب خوارزمية التحسين القائمة على التعلم، تم اقتراح إستراتيجية جديدة لعملية اختيار أزواج الطلاب، بناءًا على درجة كل طالب أثناء عملية التحسين. يتم تقييم معدل التقارب وكفاءة الخوارزمية المقترحة من خلال استعمال العديد من دوال الإختبار المعروفة. تُستخدم هذه الخوارزمية لحل مشكلة تحسين التحكم التنبئي غير الخطي.

في وحدة التحكم PID التكيفية ذات للشبكة العصبية المقترحة ، يتم استخدام شبكة عصبية متعددة الطبقات لتحديد قيم معاملات وحدة تحكم PID التقليدية. تم تطوير خوارزمية تعديل المعاملات باستخدام طريقة الانتشار العكسي. تم تحليل وحدة التحكم المقترحة ومقارنتها مع العديد من وحدات التحكم المختلفة من خلال المحاكاة الحاسوبية والدراسة التجريبية.

وحدة التحكم المقترحة الثانية تسمى وحدة التحكم PID التكيفية ذات الشبكة العصبية من نوع سلسلة فورييه. في هذا العمل، نظرًا لبنيته البسيطة وخصائصه الجذابة للغاية تُستخدم الشبكة العصبية من نوع سلسلة فورييه لضبط معاملات وحدة التحكم PID. لتقييم فعالية وحدة التحكم المقترحة، يتم التطرق إلى التحكم في ذراع الروبوت ذو ثلاثة درجات للحرية ويتم إجراء مقارنة، باستخدام العديد من خوارزميات التحكم.

يتعلق العمل الثالث المقترح با لتحكم التنبئي غير الخطي بوجود قيود على المتغيرات و باستخدام الشبكات العصبية و طريقة التحسين القائمة على التعلم. في وحدة التحكم هذه، يتم استخدام شبكة عصبية متعددة الطبقات للتنبؤ بالمخرجات المستقبلية للنظام، ويتم حل مشكلة التحسين للتحكم التنبئي باستخدام عدة طرق لإستراتيجية التحسين القائمة على التعلم (ETLBO, ITLBO, TLBO). لإثبات فعالية خوارزميات التحكم المقترحة، تم التطرق إلى التحكم في نموذج مفاعل مستمر يحرك بشكل مثالي، ونموذج ذراع الروبوت ذو درجتان للحرية، وتم إجراء مقارنة باستخدام عدة خوارزميات للتحكم.

الكلمات المفتاحية: تحكم ذكي غير خطي، شبكة عصبية اصطناعية، سلسلة فورييه، وحدة تحكم PID، تحكم تنبئي.

ABSTRACT

The main purpose of this work is to develop efficient, simple, and robust control algorithms for a large class of nonlinear systems. The idea is to improve the performance of the well known and popular controllers, namely the PID controller and the predictive control, using artificial intelligence tools, such as neural networks, fuzzy logic and meta heuristic optimization methods. The study of meta heuristic optimization methods has allowed to determine the appropriate optimization method that can be used in real time control applications and give good performance. In fact, this study has also allowed proposing an improvement to the teaching learning based optimization algorithm. Furthermore, the carried out research work has allowed proposing several control algorithms, namely the adaptive neural network PID controller, the adaptive Fourier series neural network PID controller, and the neural network model predictive control using the teaching learning based optimization method.

In order to improve the convergence rate of the teaching learning based optimization algorithm, a new strategy to the selecting process of the students' pairs, based on the grade of each student during the optimization process, is proposed. The convergence rate and the efficiency of the modified algorithm are assessed by considering several well-known benchmark functions. This algorithm is used to solve the optimization problem of nonlinear predictive control.

In the proposed adaptive neural network PID controller, a multilayer percepron neural network is used to online determine the gain values of the conventional PID controller. The adaptation algorithm is developed using the back propagation method. The proposed controller is analyzed and compared with several different controllers through computer simulation and experimental study.

The second proposed controller is called adaptive Fourier series neural networks PID controller. In this work, due to its simple architecture and very attractive proprieties, the Fourier series neural network is used to online adjust the parameters of the PID controller. To assess the effectiveness of the proposed controller, the control of a 3-DOF robot arm manipulator is considered and a comparative study, using several control algorithms, is carried out.

The third work concerns the constrained nonlinear predictive control using neural networks and teaching learning based optimization. In this work, a feed forward multilayer neural network is used to predict the future outputs of the system, and the optimization problem of predictive control is resolved using different versions of the teaching learning based optimization strategy; namely the TLBO algorithm, the Improved TLBO (ITLBO) and the enhanced TLBO (ETLBO). To demonstrate the effectiveness of the proposed control algorithms, the control of the model of the continuous stirred tank rector, and the 2-DOF manipulator robot model, is considered and a comparative study, using several control algorithms, is carried out.

Keywords: intelligent control, artificial neural network, Fourier series, PID controller, predictive control, TLBO, meta-heuristic.

RESUME

L'objectif principal de ce travail est de développer des algorithmes de contrôle efficaces, simples et robustes et qui peuvent s'appliquer sur une large classe de systèmes non linéaires. L'idée est d'améliorer les performances des méthodes de commande bien connues et populaires, à savoir le régulateur PID et la commande prédictive, en utilisant les outils de l'intelligence artificielle, tels que les réseaux de neurones, la logique floue et les méthodes d'optimisation méta heuristique. L'étude des méthodes d'optimisation méta heuristique a permis de déterminer la méthode d'optimisation appropriée qui peut être utilisée dans les applications de commande en temps réel et donner de bonnes performances. En fait, cette étude a également permis de proposer une amélioration de l'algorithme d'optimisation basé sur l'apprentissage par enseignement (teaching learning based optimization). De plus, les travaux de recherche menés ont permis de proposer plusieurs algorithmes de commande, à savoir le régulateur adaptatif PID en utilisant un réseau neuronal, le régulateur adaptatif PID en utilisant un réseau neuronal de type série de Fourier et la commande prédictive à modèle neuronal en utilisant la méthode d'optimisation basée sur l'apprentissage par l'enseignement.

Afin d’améliorer le taux de convergence de l’algorithme d’optimisation basé sur l’apprentissage par l’enseignement, une nouvelle stratégie de sélection des paires d’étudiants, basée sur la note de chaque étudiant au cours du processus d’optimisation, est proposée. Le taux de convergence et l'efficacité de l'algorithme modifié sont évalués en considérant plusieurs fonctions de test couramment utilisées. Cet algorithme est utilisé pour résoudre le problème d'optimisation de la commande prédictive non linéaire.

Dans la commande PID adaptative en utilisant les réseaux de neurones, un réseau de neurones multicouche est utilisé pour déterminer en ligne les valeurs des gains d'un régulateur PID conventionnel. L'algorithme d'adaptation est développé en utilisant la méthode de rétro-propagation. Le régulateur proposé est analysé et comparé, par simulation et expérimentalement, avec plusieurs régulateurs.

Le deuxième algorithme de commande proposé, est la commande PID adaptative en utilisant un réseau de neurones de type série de Fourier. Dans ce travail, en raison de son architecture simple et de ses propriétés intéressantes, un réseau neuronal de type série de Fourier est utilisé pour ajuster en ligne les paramètres du régulateur PID. Pour évaluer l'efficacité de l'algorithme de commande proposé, la commande d'un bras manipulateur à 3 degrés de liberté est envisagée et une étude comparative, en utilisant plusieurs algorithmes de commande, est réalisée.

Le troisième travail est consacré à la commande prédictive non linéaire avec contraintes à modèle neuronal et en utilisant plusieurs versions de l'algorithme d'optimisation basée sur l'apprentissage par enseignement. Dans cet algorithme, un réseau neuronal multicouche est utilisé pour prédire les sorties futures du système, et le problème d'optimisation associé est résolu en utilisant plusieurs version de la méthode d'optimisation basée sur l'apprentissage par enseignement; à savoir l'algorithme TLBO, le TLBO amélioré (ITLBO) et le ETLBO. Pour démontrer l'efficacité des algorithmes de commande proposés, la commande du modèle du réacteur continu parfaitement agité, et du modèle d'un bras manipulateur à 2 degrés de liberté, sont considérés et une étude comparative, en utilisant plusieurs algorithmes de commande, est réalisée.

Mots clés : contrôle intelligent non linéaire, réseau de neurones artificiels, série de Fourier, contrôleur PID, contrôle prédictif, TLBO, méta-heuristique.

LIST OF FIGURES

LIST OF TABLES

CONTENT

INTRODUCTION

Control theory is a field of applied mathematics that aims to control the behavior of dynamical systems. The objective is to develop a control algorithm (controller), which can drive the controlled system to a desired state by manipulating its inputs, while minimizing a given cost function and ensuring a certain level of stability. This function contains different criteria, such as: a delay, an overshoot and a steady state error. The control system can be implemented in an open or a closed loop. To generate the control signals, open-loop controllers only require the desired reference trajectories, while the closed loop controllers (feedback controllers) require, in addition to the desired reference trajectories, at least one feedback signal from the controlled system. Each control type has its advantages and is best suitable to particular types of control. However, the control action in closed loop control systems is based on the outputs; these controllers have the ability to self-correct, while an open loop controller has not this ability and the outputs have no effect on the control action.

Control theory has been introduced for the first time in 1868, by the physicist James Maxwell [1], where, a centrifugal governor (controller) was used to control a windmill velocity, and the impact of the self-oscillation phenomenon on the system stability has been analyzed. After that, Edward John Routh used Maxwell's results to advance the control theory [2–4]. In 1895, using differential equations, Adolf Hurwitz introduced the Routh-Hurwitz theorem [5,6], which analyzes the stability of linear systems without solving them. In December 1903, the Wright brothers made their first successful controlled flight as they installed a mechanical controller at the back of their plane to control its trajectory. In 1922, the famous Proportional Integrate Derivative (PID) controller has been introduced by Nicolas Minorsky [7]. Beside the PID regulator, several linear controllers have been developed. Such as: the Full State Feedback (FSF) or pole placement [8], the Linear Quadratic Regulator (LQR) [9], the Linear Quadratic Gaussian (LQG) [10,11], the Model Algorithmic Control (MAC) [12], the Dynamic Matrix Control (DMC) [13], … etc.

Linear controllers can be used to control nonlinear systems, by approximating their dynamic behavior around the operating conditions with linear models. Indeed, linear controllers have been found to give satisfactory performance in many practical applications [14–21]. However, a severe degradation in the control performance of industrial systems, which have become more complex and highly nonlinear, can occur when using linear controllers. In fact, this degradation can be observed when the operating conditions deviate from the steady states around which the model has been linearized. Hence, to improve the control performance of nonlinear systems, nonlinear control methods should be investigated. Indeed, a lot of attention was given to nonlinear control systems, and several algorithms, such as the nonlinear PID controller [22–25], the sliding mode control [26–28], the bang-bang control [29,30], the Nonlinear Model Predictive Control (NMPC) [31,32], the Nonlinear Generalized Predictive Control (NGPC) [33,34], were proposed. However, designing a nonlinear controller that ensures the stability of the closed-loop system is often a difficult, a complex and a time consuming task. Contrary to the theory of linear systems, there is no general control method that can be used with a large class of nonlinear systems, and most of the developed techniques are limited to very particular classes of these systems.

One of the solutions that were investigated to get around the complexity of controlling nonlinear systems is to develop intelligent control techniques having the ability of learning, adaptation and decision-making. Indeed, biological systems and, in particular, humans provide an example. Thus, an imitation, even partial, of their capacities provides an important contribution to the theory of systems. This imitation can be done by analyzing and understanding the structural and functional aspects of these natural systems. Then, using graphical representations and mathematics, model the identified components of intelligence. Previous work in this context has led to the advent of powerful tools such as Neural Networks (NN), Fuzzy Logic (FL) and meta-heuristic optimization algorithms. The modeling and control methods using Artificial Intelligence (AI) tools, such as neural networks, fuzzy logic, machine learning and meta-heuristic optimization, are grouped under the name intelligent control. Since the introduction of the concept of neural networks and fuzzy set theory, several methods of nonlinear systems modeling and control using neural networks and fuzzy logic have been proposed. Hybrid control methods that combine neural networks and fuzzy logic, neural networks and meta-heuristic optimization, fuzzy logic and meta-heuristic optimization, and conventional linear control methods with artificial intelligence have also been considered by several researchers.

Several structures of Neural Network Controllers (NNCs) have been proposed. In 1988, Hiroyuki et al. proposed an architecture based on the feedback error learning neural network to control the angular positions of a robot manipulator [35]. In this architecture, two neural networks were used; the first one simulates the dynamics of the manipulator, and the second one calculates the torque signal that reduces the error between the desired and the real angular positions of the manipulator. In the same year, another feedback NNC have been proposed [36], where a novel approach to emulate the system using neural networks was developed. One year later, a novel NNC method that control unknown dynamic systems was presented [37]. This NNC is an adaptive controller for nonlinear time-invariant systems. Unlike the NNCs presented before, this NNC requires only one neural network. Except the system order, the dynamics of the system are considered unknown. After that several applications and strategies based on this NNC method, such as the Gaussian networks for direct adaptive control [38], the Adaptive control of nonlinear systems using neural networks [39] and the nonlinear self-tuning adaptive control [40], were considered. A dynamic manipulator controller, where a back-propagation NN is used to predict the torque of each joint based on the values of the desired angles position, was proposed [41]. The NN was implemented in parallel with a Proportional Derivative (PD) controller. It has been shown, through various simulations on the PUMA 560 robot, that this NNC has fast convergence rate. Nineteen ninety was the year in which the NNC methodology captured the most attention of researchers [42]. Miller et al. [43] have developed a new architecture, called Cerebellar Model Arithmetic Computer (CMAC), which was implemented to control a robot manipulator. The CMAS architecture is based on the Least Mean Square (LMS) training process; it has plenty of advantages, such as fast computation, incremental training, output superposition and simple hardware realization. A comparative study between the CMAS, the self-tuning regulator and the Lyapunov based model reference adaptive controller, was given in [44]. The obtained results showed the superiority of the CMAS against linear and nonlinear systems with and without noise. A novel NNC approach, that uses a new unsupervised learning algorithm was developed [45]. It has been shown that this NNC can give good performance against complex nonlinear systems. In the same year, an adaptive model-based NNC for robot manipulator, was proposed to control the PUMA 560 robot [46]. This method uses two NN for payload estimation, during high-speed motion, and for error compensator. After that, a similar controller called Robust Model-Based NNC, that is based on the pseudo continuous-time analog quantitative feedback theory, was developed [47]. Several other NNCs architectures, such as the Nonlinear NNC for dynamic system [48], the NN compensator for uncertainties of robotic manipulators [49], the NN for self-learning control systems [50], the adaptive control using NNs [51], were developed in 1990.

In 1992, using neural network, K.S. Narendra et al. have proposed a control strategy to deal with structural failures of the system [52]. After that, the same authors proposed a globally stable adaptive controller for a restricted class of nonlinear systems in the presence of uncertainties [53]. In the year 2000, Zhang et al. presented a new NNC architecture [54]. In this work a smooth and singularity-free adaptive controller for first order systems, for which the stability of the closed-loop system is guaranteed, was designed. Two different back stepping NNC strategies for a class of nonlinear systems were proposed in [55]. Using the dynamic surface control, Wang and Huang have proposed an NNC that guarantees the closed-loop stability and gives a small tracking error [56]. In 2008, a generalized HJB (Hamilton Jaccobi Bellman) formulation based on NNC for Nonlinear Discrete-Time Systems (NDTS) was proposed [57]. In the same year, an adaptive NNC for a class of pure-feedback NDTS was presented [58]. In the last decade, several NNC strategies, such as the adaptive NNC for output feedback nonlinear systems using a barrier Lyapunov function [59], the adaptive NN decentralized back stepping output-feedback control for nonlinear large scale systems with time delays [60], the adaptive control with high-order NN [61], the NN based adaptive control for a class of uncertain nonlinear stochastic systems [62], the dynamic surface control using NN [63], the adaptive NNC of an uncertain robot with full state constraints [64], the NNC based adaptive learning design for nonlinear systems With full state constraints [65], the adaptive NN finite time output feedback control of quantized nonlinear systems [66], the improved radial basis function NNC strategy [67], and the NNC of robot manipulators and nonlinear systems [68], have been developed.

Due to the success achieved using NNC against nonlinear systems, several hybrid controllers, based on NN, have been proposed. Among these controllers, the neuro-fuzzy controllers have received a lot of attention due to its high performance against nonlinear systems. Until 1991, there was no systematic procedure for designing an FLC, at the time, the approach was to define the membership functions and the rules by studying an already existing human system or controller. However, Lin et al. proposed the Integrated Neural Network based Fuzzy Logic Control (INNFLC), where an NN was organized to mimic the FLC by integrating the learning capability of the NN with the FLC system [69]. The resulting controller is a Multi Layer Perceptron (MLP) that has three hidden layers. Where, the nodes of the first layer and the third layer represent the membership functions of the FLC. The second layer integrates the rule engine into the network. After that, similar architectures were proposed, such as the fuzzy modeling using generalized neural networks and Kalman filter algorithm [70] and the rule extraction using generalized neural networks [71]. In 1993, the famous Adaptive Network based Fuzzy Inference System (ANFIS) was presented [72]. The ANFIS is based on human knowledge and stipulated input-output data pairs. Several application of the ANFIS in system control have been considered [73–77]. Other ANFIS variants can be found in the literature [78–80].

Model Predictive Control (MPC) is one of the most important control strategies, it is a modern and successful technique that is characterized by its ability to control constrained multivariable system. Several predictive control algorithms, based on linear models, were developed [12,13,81–86]. The success of the MPC strategy is related to its capability of handling different types of constraints, and its ability to handle multivariable systems, systems with no-minimum phase behavior and systems with variable or unknown time delays. Although, linear MPC techniques give satisfactory performance in many practical applications [87–89], in the case of highly nonlinear process, severe degradation in the control performance can be observed. To ensure higher performance, MPC methods that use a nonlinear prediction model must be investigated. In fact, a lot of attention was given to Nonlinear MPC (NMPC), and several control techniques were proposed [90–92]. The main difficulties in designing NMPC are obtaining an adequate nonlinear model for the system to be controlled, and online solving the nonlinear and the non-convex optimization problem. Obviously, the efficiency and the computational requirement of the controller depend extremely on the accuracy and the simplicity of the used model. Among the various nonlinear models developed and used in predictive control, we can find Volterra series [93–95], Fuzzy models [91,96–98], and NN models [90,99,100].

Solving the non-convex optimization problem of nonlinear constrained predictive control is difficult and time-consuming task. Since an analytical solution cannot be obtained, several suboptimal methods were used to deal with the optimization problem [101–103]. These suboptimal methods can be classified into three categories: the linearization of the nonlinear model, the use of a particular structured model that gives a free response and a forced response, and the use of numerical approaches that are nonlinear optimization techniques. Numerical approaches can be classified into two classes: the first class join the deterministic numerical approaches such the sequential quadratic programing and nonlinear interior point method. These methods handle the optimization problem by solving series of linear sub-problems. However, they are known for their sensitivity to initial conditions and cannot be used with models where the details of the derivatives are unknown. The second class gathers stochastic numerical approaches such as meta-heuristic optimization algorithms. Meta-heuristic algorithms could mitigate the non-convex and nonlinear optimization problem of predictive control. Indeed, Meta-heuristic algorithms are easy to implement, have good performance, and could locate adequate solutions in a reasonable time. Different meta-heuristic algorithms, such as Genetic Algorithms (GA) [104,105], Particle Swarm Optimization (PSO) [90,106,107], Artificial Bee Colony (ABC) [96,97], Evolutionary Algorithm (EA) [108], and Teaching Learning Based Optimization (TLBO) [109,110], can be found in the literature. In fact, a lot of researches have been conducted on solving the NMPC optimization problem using meta-heuristic algorithms [90,96,97,105,107].

The basic objective of this doctoral thesis is to develop control algorithms for nonlinear systems using the tools of artificial intelligence. The aim is to design control methods that are:

- Simple to guarantee their implementation in real time.
- Efficient and give good control performance with a large class of nonlinear systems.
- Adaptive to compensate parameters variation and external disturbances.
- Robust against modeling errors and external disturbances.

This interest is justified by the fact that the physical systems are, in a very large majority, nonlinear subject to modeling errors and parametric uncertainties, as well as to external disturbances. The ability of neural networks to approximate uniformly continuous functions has been proven in several papers. Neural networks, with the ability to approximate a large class of nonlinear functions, provide a canonical and feasible structure for the representation of non-dynamic systems. Meta-heuristic algorithms are easy to implement, have good performance, and could converge to global solutions in a reasonable time.

The research directions explored in this thesis are:

- Artificial intelligence based adaptive nonlinear PID controller: the aim is to use different architectures of neural networks to online obtain the PID gains,

- Teaching learning based optimization: develop an improved version of the TLBO algorithm that can be used to online solve the optimization problem of predictive control,
- Neural network based constrained nonlinear predictive control: the objective is to develop an efficient constrained nonlinear predictive control algorithm using a multilayer feed forward neural network and the teaching learning based optimization.

Single-input single-output and multi-input multi-output nonlinear systems are considered in this thesis. To highlight the performance of the proposed control algorithms, a comparative study, using simulation and experimental setup, is carried out and the control of the following systems is considered:

- The model of the continuous stirred tank reactor,
- The induction machine,
- The three degrees robot manipulator.

The thesis is organized as follows:

In the first chapter, some of the AI based control algorithms are presented and discussed. NN controllers and fuzzy logic control algorithms are given. Furthermore, some hybrid neural network fuzzy logic controllers are presented and the AI based MPC techniques are also described.

In the second chapter, a general description of meta-heuristic algorithms is given, then a detailed description of three meta-heuristic algorithms, namely the particle swarm optimization, the teaching learning based optimization and the Improved TLBO (I-TLBO) is introduced. The proposed enhanced TLBO (ETLBO) algorithm is presented and compared to several meta-heuristic algorithms using some well-known benchmark functions.

In the third chapter, three versions of the AI based PID controller are given. The Adaptive NNPID (ANNPID), the proposed Adaptive Fourier Series Neural Network PID (AFSNNPID) and the PSO based PID are given. To assess the effectiveness of the ANNPID, the AFSNNPID and the PSO based PID controller, the control of the Continuous Stirred Tank Reactor (CSTR) and the 3-DOF robot arm manipulator, through simulation and experimental study, are considered.

In the fourth chapter, three proposed AI based NMPC controllers are presented and evaluated by considering the control of the MIMO robot arm manipulator.

The thesis finishes with concluding remarks and some prospects for the future works.

CHAPTER 1

AI-BASED CONTROL ALGORITHMS

1. Introduction

This chapter introduces a review of some AI based control methods, such as the Neural Network Control (NNC), the Fuzzy Logic Control (FLC), the Fuzzy Neural Network Control (FNNC) and the AI-based Nonlinear Model Predictive Control (AI-NMPC).

2. Neural network control

Due to their learning and generalization capabilities and their parallelism, neural networks have achieved a great success in many areas. They have been successfully used in many applications, such as classification, noise filtering, system modeling and control … etc. One of the fields where neural networks have received increasing interest is that of systems modeling and control. Indeed, significant research in this field has been carried out and several control applications based on neural networks have been developed. Most neural control methods, that were proposed, can be classified in two main categories:

- Direct control: in this class, the controller is a neural network that delivers the control signals.
- Indirect control: in this class, the design of the control law is based on the use of a neural model of the system to be controlled.

In the rest of this section, some of the common neural control structures are presented.

2.1. Neural control structures

Generally, a neural control structure contains one or more neural networks. Each neural network is used to perform one of the following specified tasks:

- Generation of control signals: the neural network is used to compute the control signals that minimize the control error given by the difference between the system outputs and the reference trajectories. The inputs of this neural network can be the actual and the past values of the reference trajectories and the actual and the past values of the system outputs or the actual and the past values of the control error (figure 1.1 (a) and (b)).
- System emulation: In this case, the neural network is trained and used as an emulator of the dynamics of the controlled system. Any of the common neural network architectures, such as feed forward neural networks or recurrent neural networks, can be used. In general, the inputs to this neural network are the actual values of the control signals and the system outputs and their delayed values (figure 1.2).
- Reference model: the neural network is used to modify the desired reference trajectories for the purpose of improving the control performance. This neural network receives the actual and the past values of all or some reference trajectories as inputs and generates a modified reference trajectory (figure 1.3).

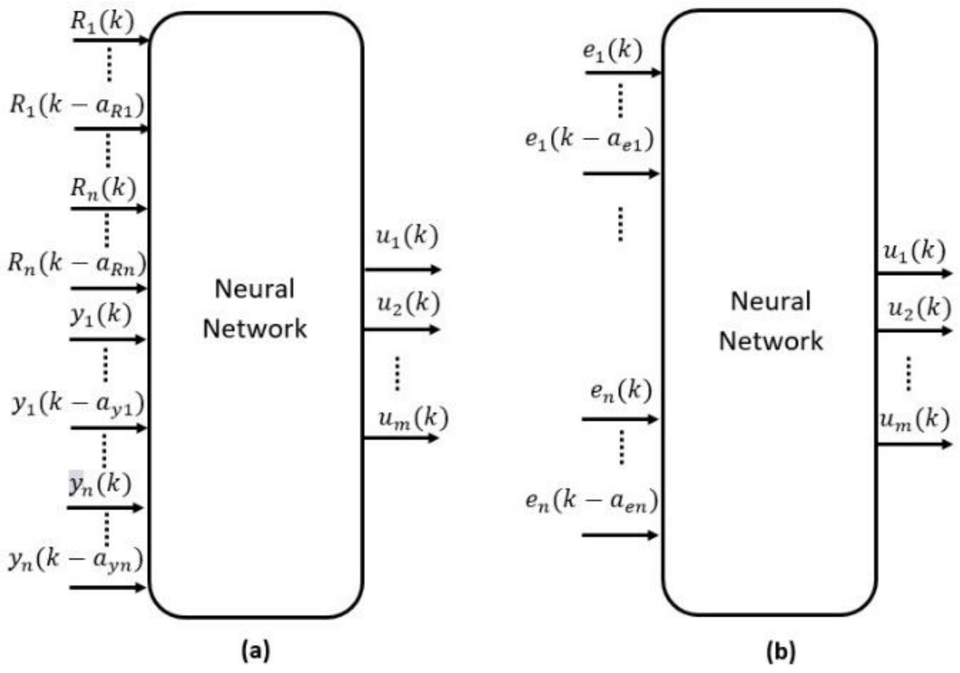


Figure 1.1

Figure 1. 1 : Inputs and outputs of the neural network controllers.

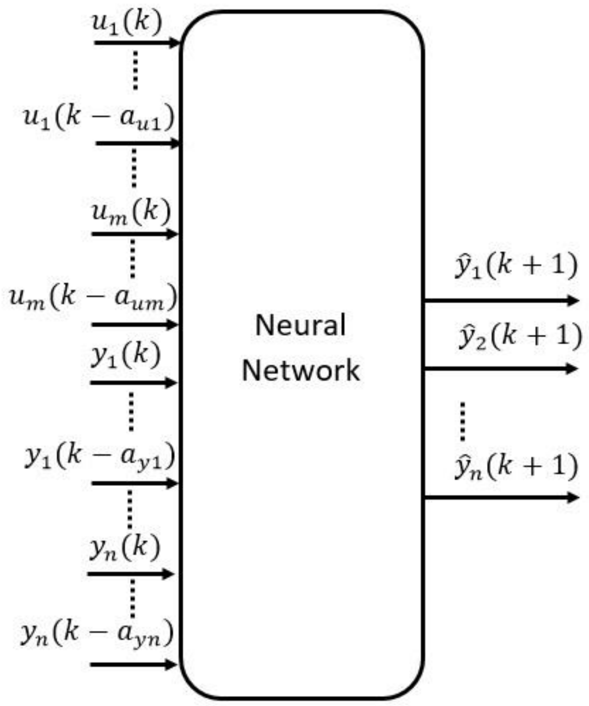


Figure 1. 2 : Inputs and outputs of the emulator.

Several architectures of NNC can be built according to the number of the used NNs in the control block diagram and the task given to each NN. Hence, some types of the NNC are presented in the following subsections.

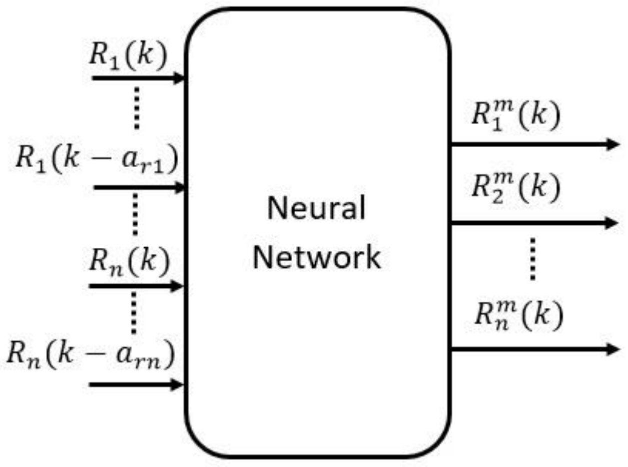


Figure 1. 3 : Neural network as a model reference.

2.1.1. <u>Direct control using the inverse dynamic model (general learning)</u>

Direct control using the inverse dynamic model [111–114] is also known with the general learning architecture. It is built in two successive stages. First, using a neural network, the inverse dynamic model of the controlled system is constructed (figure 1.4.a). The neural network is trained using the backpropagation algorithm in order to produce an output as close as possible to the control signal. In a second step, the trained neural network is implemented for the purpose of controlling the system in an open-loop configuration (figure 1.4.b). Several variants of this control strategy can be found in the control literature. The main difference between these variants is the architecture of the used neural network.

Often, it is not easy to obtain an inverse model to all dynamic systems. Even if the inverse dynamics has been successfully modeled, a small disturbance leads to poor control performance. To deal with this problem, a new neural network is added to the control scheme to online update the weights of the inverse model controller (figure 1.4.c).

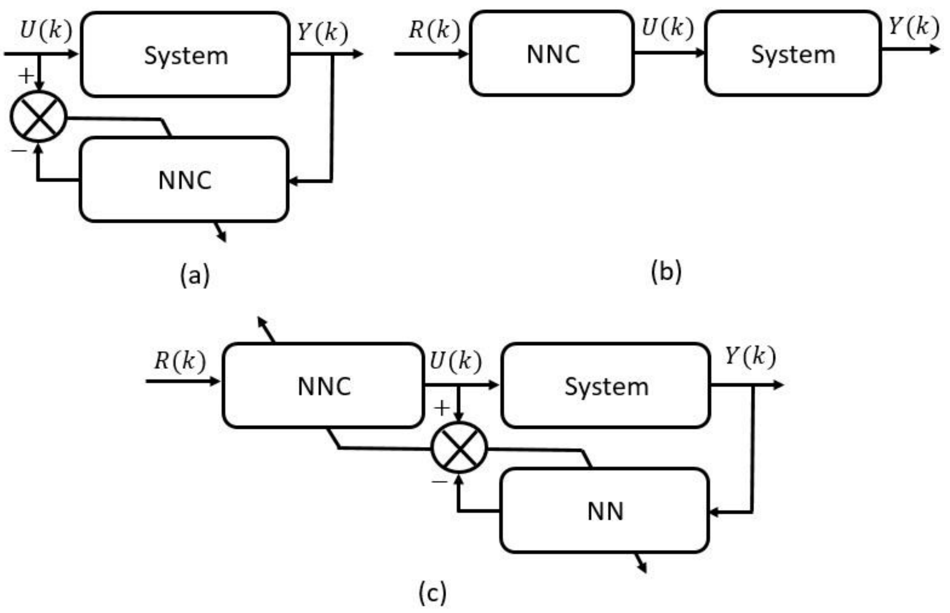


Figure 1. 4 : Direct control using the inverse dynamic model.

2.1.2. Supervised control

The neural network based supervised control strategy (figure 1.5) aims to emulate and substitute an already existing controller (for example a human operator) using a neural network [115,116].

This method is useful in many cases, such as:

- The original controller, which was used to train the neural network, can be complex and hard to implement.
- The computation process of the original controller includes high level operators (e.g. integral) that requires an important computing time.
- The original controller can be a human operator who cannot be permanently assigned to the control task.

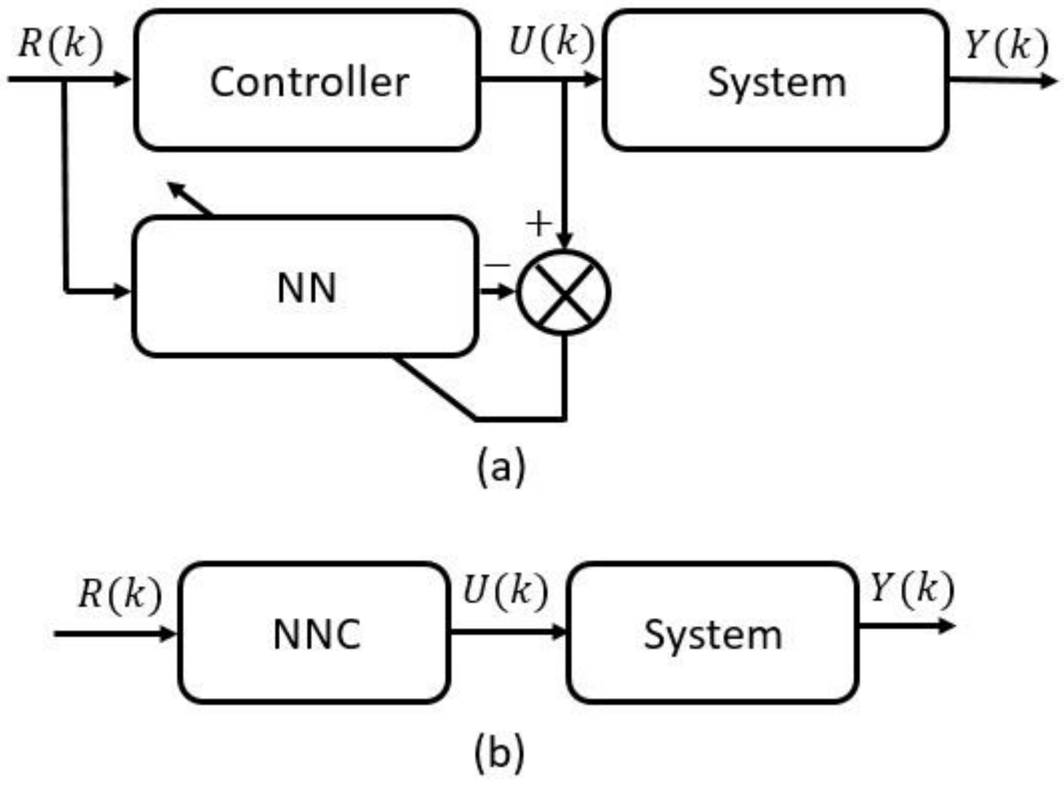


Figure 1. 5 : Supervised Control strategy: (a) training the controller, (b) substitution of the existing controller.

2.1.3. Direct control using the inverse dynamic model (specialized learning)

The control block diagram of this strategy is given in figure 1.6, where the neural network drives directly the controlled system [117]. The error signal between the system output and the desired reference trajectory is used to adjust the weights of the neural network controller.

Using this strategy, most of the drawbacks of the general learning strategy are omitted. The offline adaptation can be ignored, the controller is online adapted and the control objective is directly handled. However, this strategy requires a priori knowledge of the controlled system to train and adapt the weights of the neural network controller. In this control strategy, a neural network that emulates the system dynamic is used in the neural network controller training process.

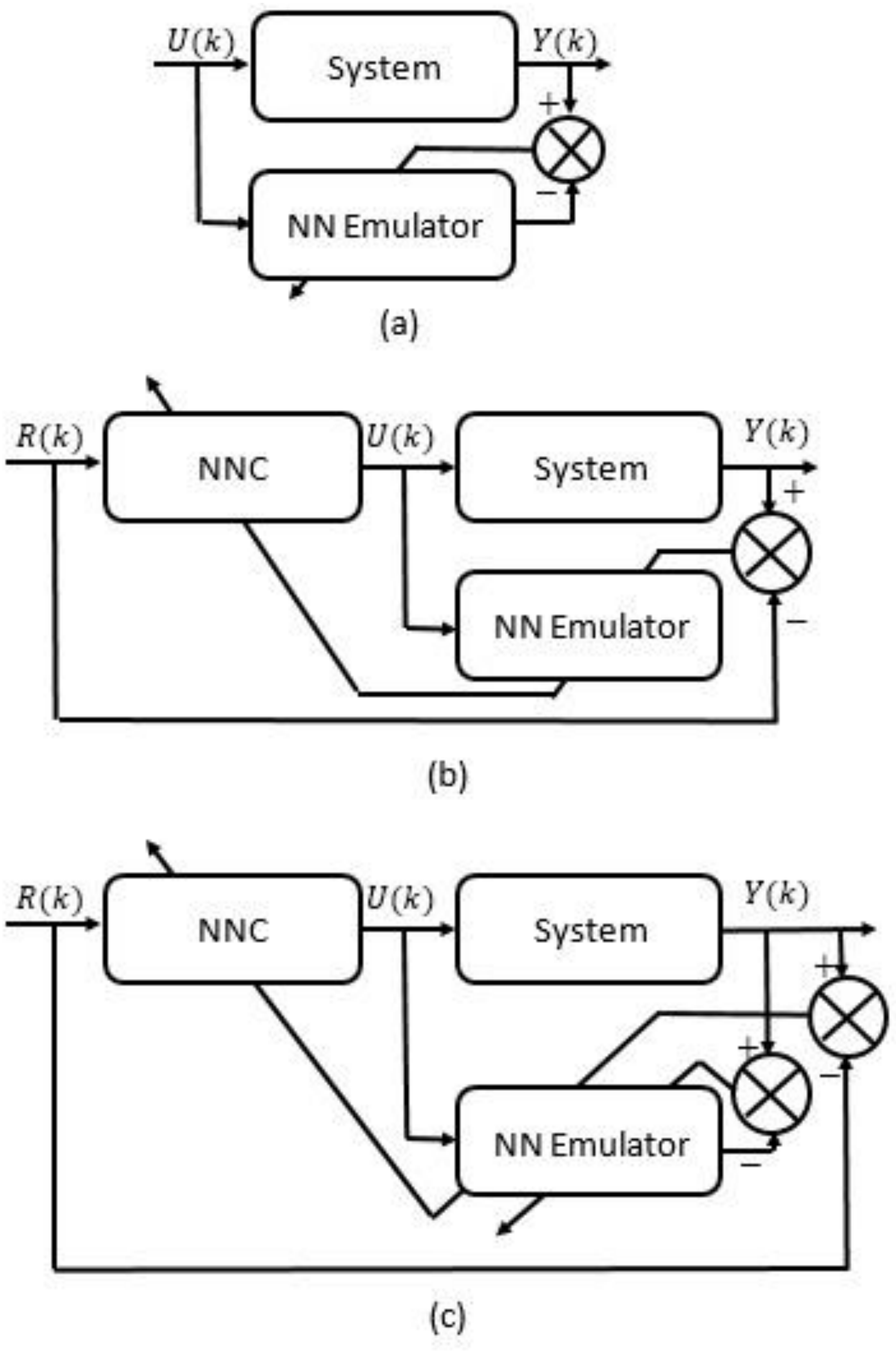


Figure 1. 6 : Direct control using the inverse dynamic model (specialized learning).

The specialized learning control approach can be achieved in two successive phases: an identification phase for the neural network emulator (figure 1.6.a), followed by the neural network controller learning phase (figure 1.6.b). In fact, both phases can be merged into one step as it is indicated in (figure 1.6.c).

2.1.4. Model reference adaptive control

In this strategy, a reference model is used to specify the desired behavior of the controlled system, and the neural network controller is used to force the output of the system to follow that of the reference model [118,119]. Based on this principle, two control structures have been proposed; the direct (figure 1.7 (a)) and the indirect methods (figure 1.7 (b)).

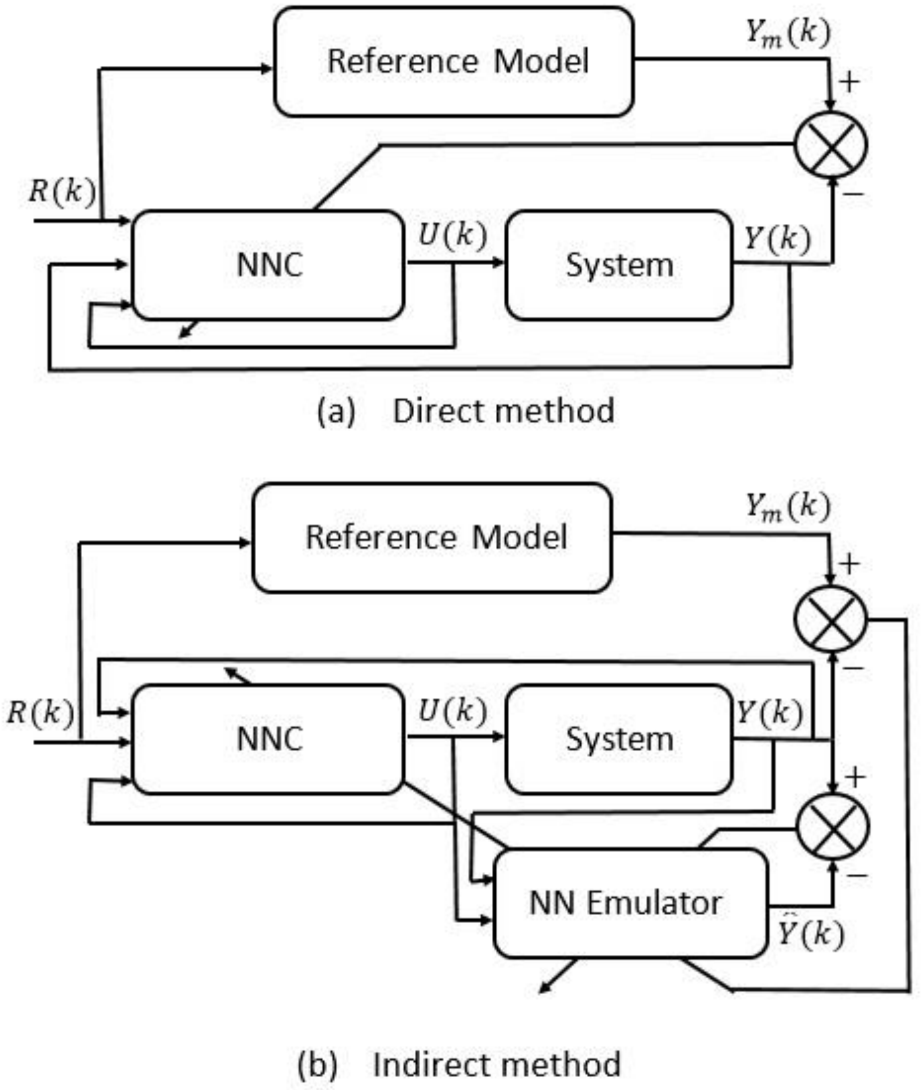


Figure 1. 7 : Model reference adaptive control.

In the direct Model Reference Adaptive Control (MRAC) strategy, the weights of the neural network controller are directly adjusted to reduce the error between the system output and the reference model output. Once the learning phase is completed, the neural network controller generates the control signal so that the output of the system follows that of the reference model. Since the system is placed between the neural network controller and the output error, using the back-propagation algorithm, the system Jacobian is required to directly adapt the weights of the neural network controller. To avoid this problem, the indirect MRAC structure has been developed.

The indirect MRAC structure uses a first neural network to identify the direct dynamics of the controlled system (emulator) and a second one to derive the control signal (neural network controller). The weights of the emulator can be adapted online to continuously follow the dynamic behavior of the system and those of the neural network controller are adapted by back propagating the control error throw the emulator.

2.1.5. Other approaches

Several other neural network controllers, which are modified versions of the previous approaches, were developed and used in many control applications. Among the most interesting techniques, we mention the following three strategies:

The first one (figure 1.8.(a)) is the neural network controller proposed in [120,121]. In this control method, a neural network is used to generate a modified reference trajectory and a classical controller, such as the PID controller, or other advanced controller is used to drive the system.

In the control structure given by figure 1.8 (b), an optimal linear controller is combined with a neural network controller [122]. This strategy is based on the principle of compensating the uncertainties that are not considered during the design of the linear controller by using two neural networks. The first one is placed in parallel with the controlled system to model the nonlinearities of the system, and the second one is placed in parallel with the linear controller to compensate the effect of the uncertainties by adding a variation $\Delta U(k)$ to the linear control signal $U(k)$.

The control strategy presented in figure 1.8 (c) uses a neural network to optimize the parameters of a classic controller [123,124]. The neural network receives the control errors and the control signals as inputs and generates the parameters of the classical controller as outputs.

3. Fuzzy Logic Control

Fuzzy Logic Control (FLC) is a relatively recent approach that easily integrates knowledge and key elements of human thought into the design of nonlinear controllers. Qualitative and heuristic knowledge, which cannot be addressed by conventional control theory, can be used for control purposes in a systematic way, using fuzzy logic concepts. The main advantages of fuzzy logic control are that it does not require an accurate mathematical model, can handle imprecise inputs and nonlinearity, and is less sensitive to external disturbances than the most nonlinear controllers. The design of fuzzy logic controllers is based on fuzzy sets theory and can be achieved according to the following steps (figure 1.9):

The first step is to define the inputs and outputs of the controller. There are no general rules to select the controller inputs; however, the states of the system to be controlled, their errors and variation errors are often used. The use of rules that are expressed using linguistic terms implies the fuzzification step to map the crisp values of inputs to suitable linguistic values. After that, using an inference engine, the rules are evaluated to obtain the fuzzy control action. A defuzzification step is then required to obtain a crisp value for the control action.

Most of fuzzy logic controllers are designed based on knowledge about the controlled system. The universes of discourse of inputs and outputs and the membership functions cannot be chosen without having some available information about the system dynamics. However, after choosing the membership functions and establishing the rules base, the control action can be easily computed. Therefore, to implement a fuzzy logic controller, a high-end processor is not required, unlike most other complex nonlinear controllers.

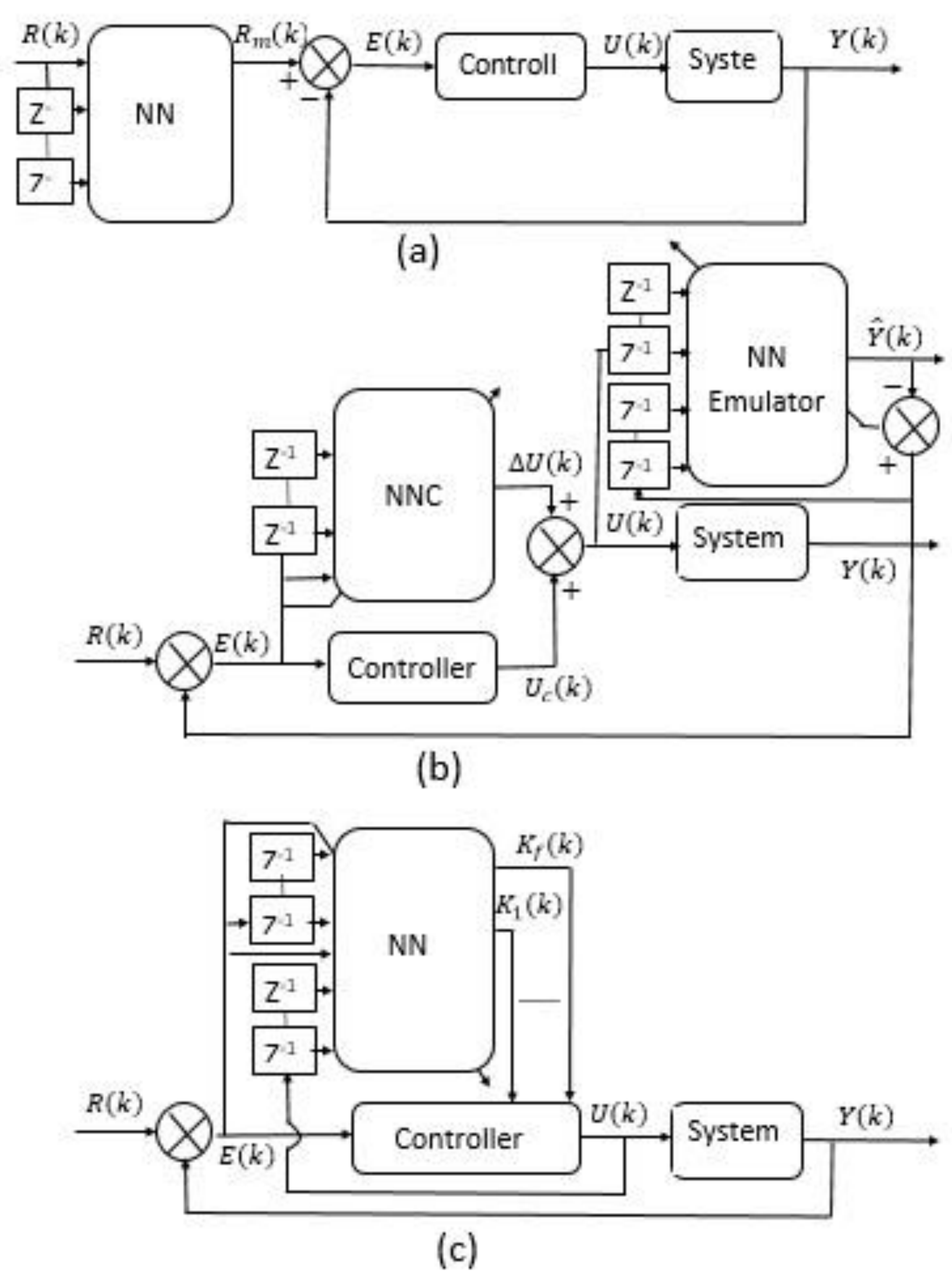


Figure 1. 8 : Other Neural Network Controllers.

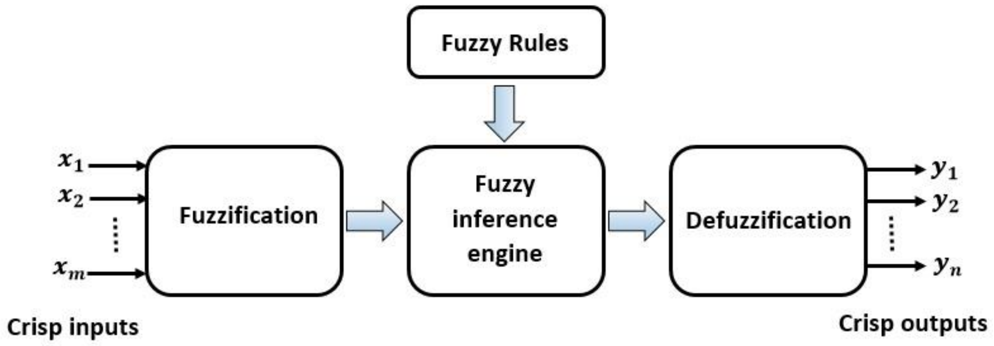


Figure 1. 9 : Typical fuzzy logic system structure.

3.1. Fuzzy Logic Controllers

Fuzzy logic controllers are used with great success in various applications. Some of these applications include controlling temperature room, anti-braking system used in vehicles, washing machines and almost all the consumer products. Over the last few decades, numerous interesting design techniques of fuzzy logic controllers were introduced and many efficient methods that deal with the robustness, the stability, and the time delay of these controllers were developed. In the following subsections, some of the well known fuzzy control methods are presented.

3.1.1. Fuzzy inverse model control

In this control strategy, a fuzzy inverse dynamic model of the controlled system is designed using the available knowledge about the system [125]. The obtained inverse model is used to control the system; it receives the reference signals as inputs and generates the control signals as outputs (figure 1.10.a). In most fuzzy inverse model controllers, in addition to the reference signals, the state vector of the controlled system is used as input of the controller (figure 1.10.b).

Constructing a fuzzy inverse model is difficult and not possible for all dynamic systems. Even if the inverse dynamics have been successfully modeled, a small disturbance leads to poor control performance. In several applications that use a fuzzy inverse model controller, an adaptation process is used to compensate the effect of the external disturbances.

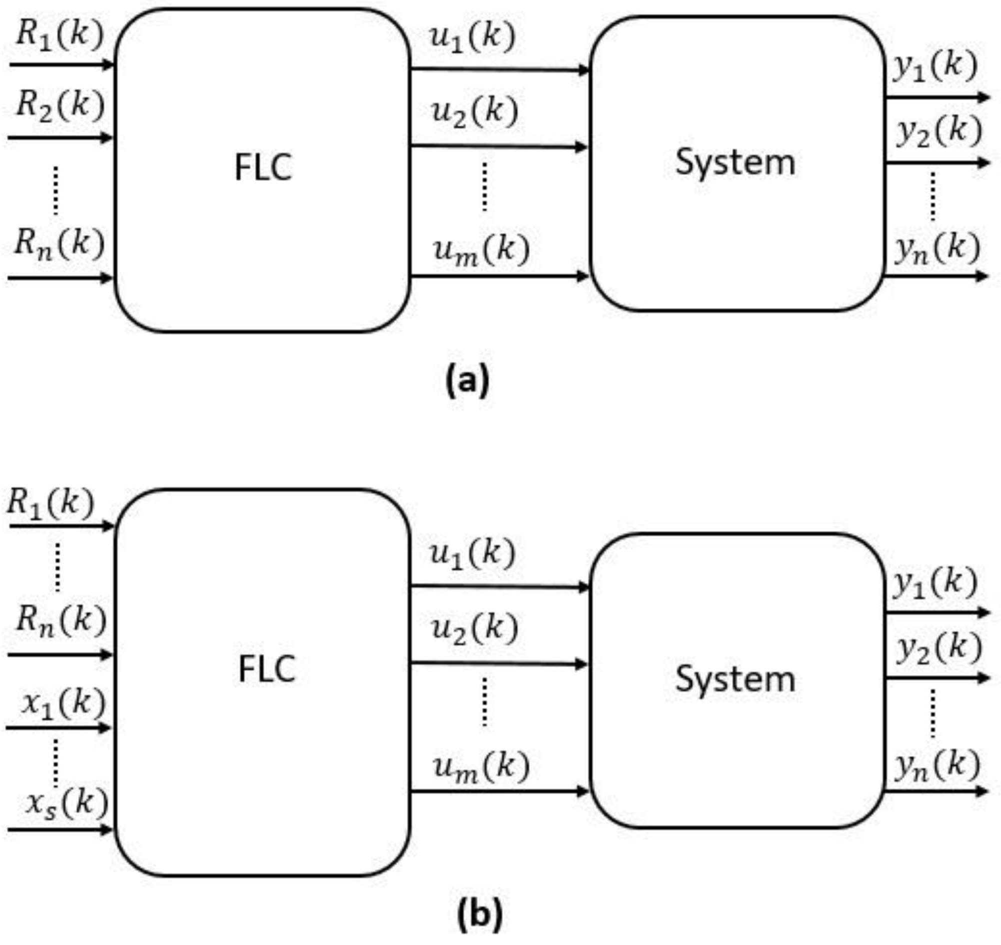


Figure 1. 10 : Direct control using fuzzy inverse dynamic model.

The main drawbacks of this control strategy are:

- The number and the complexity of rules increase with the complexity of the controlled system.
- A good knowledge of the system dynamics is required.
- Sensitivity to external disturbances.

3.1.2. Model free fuzzy control

Due to its simplicity, ease of implementation and ability to handle a large class of nonlinear systems, the model free fuzzy control [126] is used in various real-time applications. The control bloc diagram of this approach is given in figure 1.11, where the controller, usually having as inputs the error and its variations, is a fuzzy system.

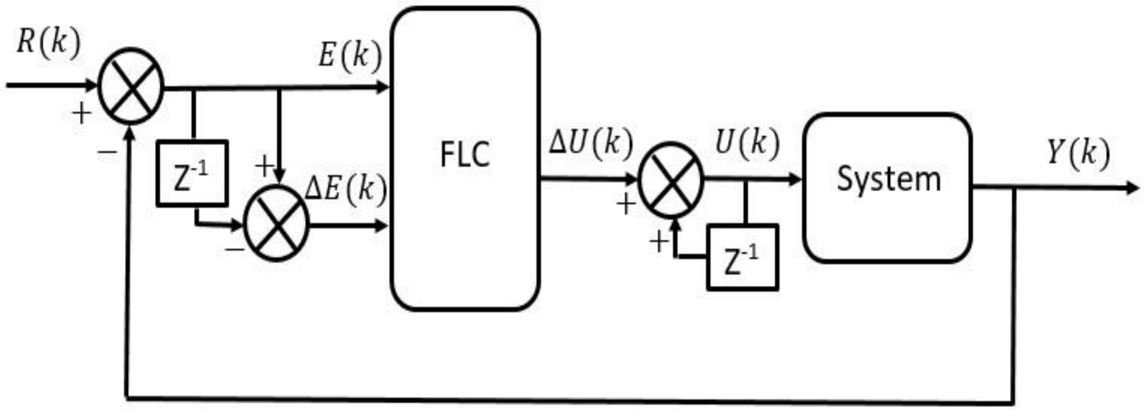


Figure 1. 11 : Control block diagram of the model free fuzzy controller.

The design steps for such controllers are summarized as follows:

3.1.2.1. Normalization and Denormalization

In this step, the universe of discourse of the inputs of the fuzzy controller is restricted to a given interval. In general, the normalized universe is identical to the real operating ranges of inputs/outputs variables, but in most applications is restricted to the interval $[-1\ 1]$. The outputs of the fuzzy controller are denormalized in order to transform the normalized values of the control signals into values belonging to their respective physical domain.

3.1.2.2. Fuzzy rules design

Constructing a fuzzy rules base has a critical role in fuzzy logic controller design and has been extensively considered. In most cases, fuzzy rules can be generated using knowledge on the system operating and by understanding of its dynamics. If each of the two input variables E and ΔE of the controller is partitioned to five fuzzy sets$(\mathrm{BN}, \mathrm{N}, \mathrm{Z}, \mathrm{P}, \mathrm{BP})$, then a total of 25 rules is required to generate a fuzzy output. For example, when the output $\Delta \mathrm{U}$ is quantized to nine fuzzy sets $(BBGN, BGN, \mathrm{BN}, \mathrm{N}, \mathrm{Z}, \mathrm{P}, \mathrm{BP}, \mathrm{BGP}, \mathrm{BBGP})$, the inference matrix can be given by table 1.1 if the system's output follows the same direction of variation of the control signal, otherwise it can be given by table 1.2.

		ΔE				
		BN	N	Z	P	BP
E	BN	$BBGN$	BGN	BN	N	Z
	N	BGN	BN	N	Z	P
	Z	BN	N	Z	P	BP
	P	N	Z	P	BP	BGP
	BP	Z	P	BP	BGP	$BBGP$

Table 1. 1 : Inference matrix for a system with direct response to the control signal.

		ΔE				
		BN	N	Z	P	BP
E	BN	$BBGP$	BGP	BP	P	Z
	N	BGP	BP	P	Z	N
	Z	BP	P	Z	N	BN
	P	P	Z	N	BN	BGN
	BP	Z	N	BN	BGN	$BBGN$

Table 1. 2 : Inference matrix for a system with inverse response to the control signal.

3.1.3. Adaptive fuzzy control

Adaptive control is based on the use of an adaptation mechanism to control partially known systems and to compensate the effects of different disturbances. Adaptive control of linear systems and some special classes of nonlinear systems has been well developed in the literature. However, developing a good adaptive controller for the majority of nonlinear systems is a challenging task.

Adaptive fuzzy control [127–129] approximates the unknown nonlinearities of the controlled system and apply the well-developed techniques of adaptive control. Adaptive fuzzy controllers can be divided into two classes:

- Direct adaptive fuzzy controllers [128]: the parameters of these controllers are tuned online in order to minimize the error between the reference model and the controlled system.
- Indirect adaptive fuzzy controllers [129]: in this case, the parameters of the controlled system are estimated using a fuzzy model and the control signal is generated based on these parameters.

3.1.4. Fuzzy PID controller

The main difficulty in designing a PID controller lies in obtaining the PID gains that give the best control performance, especially in case of high order and nonlinear systems. Obtaining the PID controller gains becomes more difficult when the system is subject to external disturbances or to variation of its parameters.

To compensate the effect of external disturbances and variation of the system parameters, several techniques using fuzzy logic have been proposed to obtain and online adjust the gains of the conventional PID controller [130,131]. Figure 1.12 (a and b) illustrates the structure of two well known fuzzy PID controllers. In order to enhance the control performance, an adaptive fuzzy system is added to the control loop. In figure 1.12.b an emulator is used to approximate the system Jacobian.

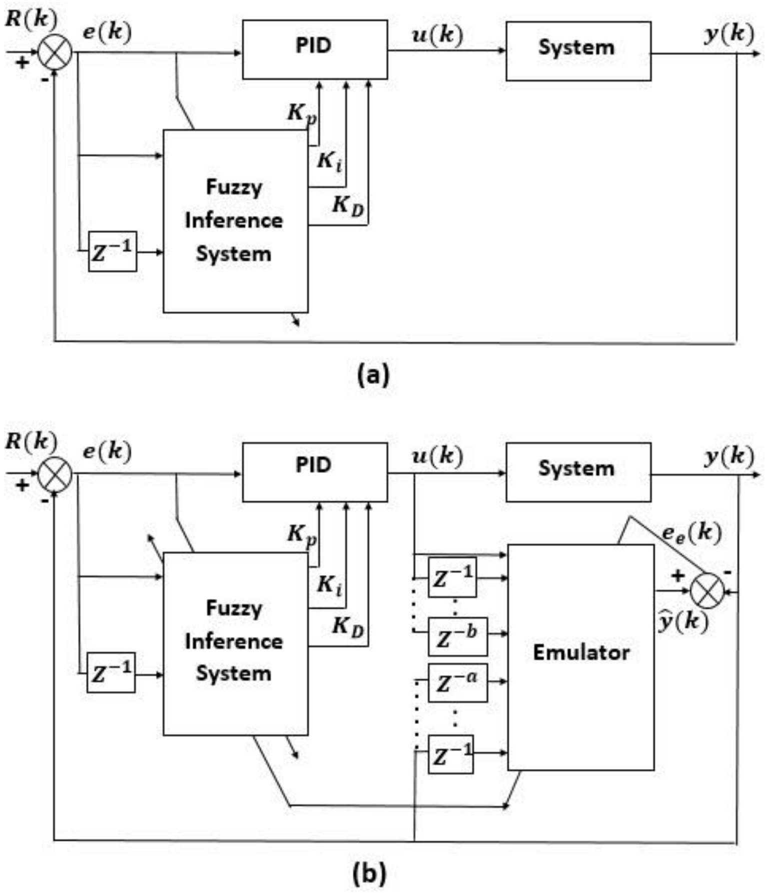


Figure 1. 12 : Fuzzy PID controller.

3.1.5. Feed-forward compensation based on fuzzy logic controller

In this architecture [132], the FLC is used in parallel with another controller (in general, a conventional controller) to improve the control performance (figure 1.13). The resulting control algorithm is a hybrid between the FLC and the conventional controller.

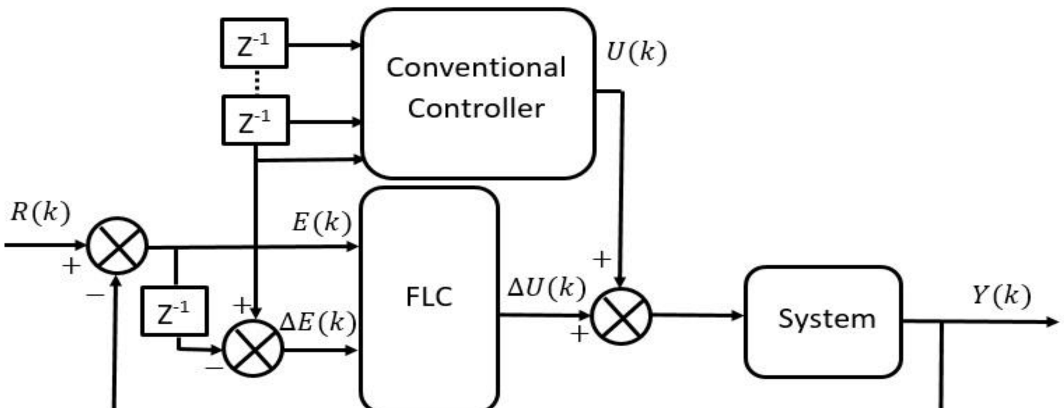


Figure 1. 13 : Fuzzy control using feed-forward compensation.

4. <u>Fuzzy neural network control</u>

A Fuzzy Neural Network Controller (FNNC) is a hybrid intelligent algorithm that combines the simplicity and the human based reasoning of the fuzzy logic and the learning ability and connectionist architecture of the neural networks. Other denominations of this approach, such as the neuro-fuzzy control and the neural network fuzzy logic control are used. The fuzzy neural network controllers have the following properties:

- A FNNC is based on the FLC, which is trained using a learning algorithm derived from neural networks.
- The FNNC architecture can be organized into three layered neural network. Each layer represents the input variables, the fuzzy rules, and the output variables. However, the FNNC architecture can be viewed as a five-layered neural network, where the two additional layers represent the fuzzy subsets.
- A FNNC is always interpreted as a process of fuzzy rules before and after learning.
- A FNNC approximates an n-dimensional function, which is partially defined by a training database.
- A FNNC can be created using the training database. The FNNC parameters can be initialized using prior knowledge about the dynamics of the controlled system.

Fuzzy neural network controllers combine the advantages of the fuzzy logic controllers and the neural network controllers in one control algorithm. The main advantages of Fuzzy neural network control algorithms are given as follows:

- Fuzzy neural network systems are universal approximators, hence the fuzzy neural network controllers are universal controllers with the ability of interpreting fuzzy rules.
- Fuzzy neural network controllers can be initialized with or without prior knowledge about the controlled system dynamics.
- Due to the fuzzy neural network controller architecture that resembles a neural network, a learning algorithm is used to adapt the controller parameters. Hence, the fuzzy neural network controllers are adaptive, which make them robust controllers with good control performances.

The FNNC can be a Mamdani or a Sugeno type. However, nowadays Sugeno-type fuzzy neural network controllers are the most used; due to the simplicity and the precision offered by the Sugeno architecture. Several Evolving Fuzzy Logic approaches can be found in [133,134]. The modern fuzzy neural network controller architectures are usually presented as a special multilayer perceptron, such as:

- The Adaptive Neuro Fuzzy Inference System (ANFIS) [72].
- Fuzzy Neural system (FuNe-I) [135].
- Fuzzy RuleNet [136].
- Generalized Approximate Reasoning-based Intelligent Control (GARIC) [137].
- NEuro Fuzzy CONtrol (NEFCON) [138].

4.1. <u>Adaptive neuro fuzzy inference system</u>

Adaptive neuro fuzzy inference system, developed in 1993 by JSR Jang [72], it is an artificial neural network based on the Sugeno-type fuzzy logic system. Its operating principle is based on a set of fuzzy IF-THEN rules that have learning capability and can approximate nonlinear functions. The architecture of an ANFIS is a five-layered network; it is given by figure 1.14.

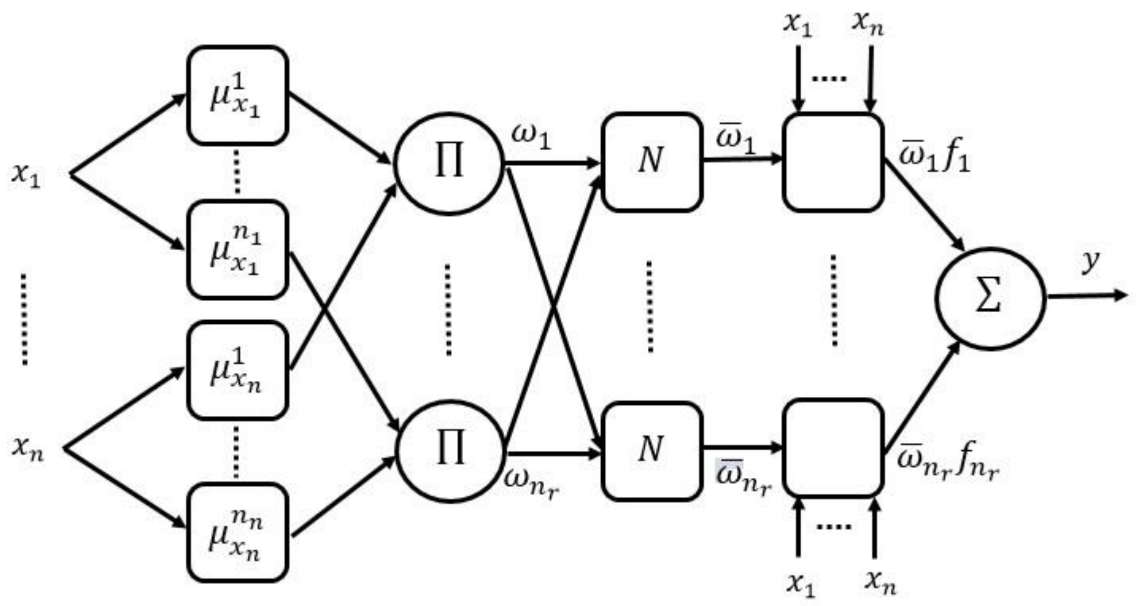


Figure 1. 14 : Architecture of adaptive neuro fuzzy inference system.

The first layer, called the fuzzification layer, takes the input values and generates the membership degrees of each input. Hence, the activation functions of the first layer' neurons are the membership functions associated to the inputs. The second layer, called the rule layer, generates the result of the premise of each rule. The product is used to calculate the AND operator. The results of the rules premises are normalized in the third layer. In the fourth layer, for each rule, the product between the normalized premise result and the rule conclusion result is calculated. Finally, in the fifth layer, the sum of the fourth layer outputs is calculated.

Generally, when using an ANFIS as a controller, the inputs are chosen as the error $e(k)$ between the desired reference trajectory and the system output, and its variation $\Delta e(k)$. The output is the increment of the control signal $\Delta u(k)$. When using the ANFIS as a controller, the system jacobian is added to the learning algorithm. Many applications of the ANFIS controller can be found in [73,74,139–141].

4.2. Fuzzy neural system

FuNe-I is a special MLP that was used to mimic a fuzzy system (figure 1.15) [135]. FuNe-I's fuzzy rules could be extracted using an input/output database and a supervised learning algorithm. This structure is used to identify the rules without a prior knowledge. Therefore, FuNe-I controllers are ideally suited for controlling systems with unknown dynamics. Furthermore, an optimization process can be implemented by tuning the parameters of the membership functions. Hence, FuNe-I controllers are adaptive and robust. Although, expert knowledge of the system dynamics is not required to create a FuNe-I controller, this knowledge can be included in the process of creating the FuNe-I control algorithm. Several control applications using FuNe-I can be found in the literature [142,143].

The gradient descent method is used to train the FuNe-I parameters, it is detailed in [135]. The structure of FuNe-I is based on positive type (if) rules and negative type (if not) rules.

Each rule is weighted by a negative or positive real number; these weights are optimized using the learning algorithm. The i^{th} FuNe-l output (y_i) is generated by calculating the sigmoid of the sum of the "r" rules weighted results ($W_{ij}K_j$) as follows:

$$y_i = sigmoid(\sum_{j=1}^{r} W_{ij}K_j) \tag{1.1}$$

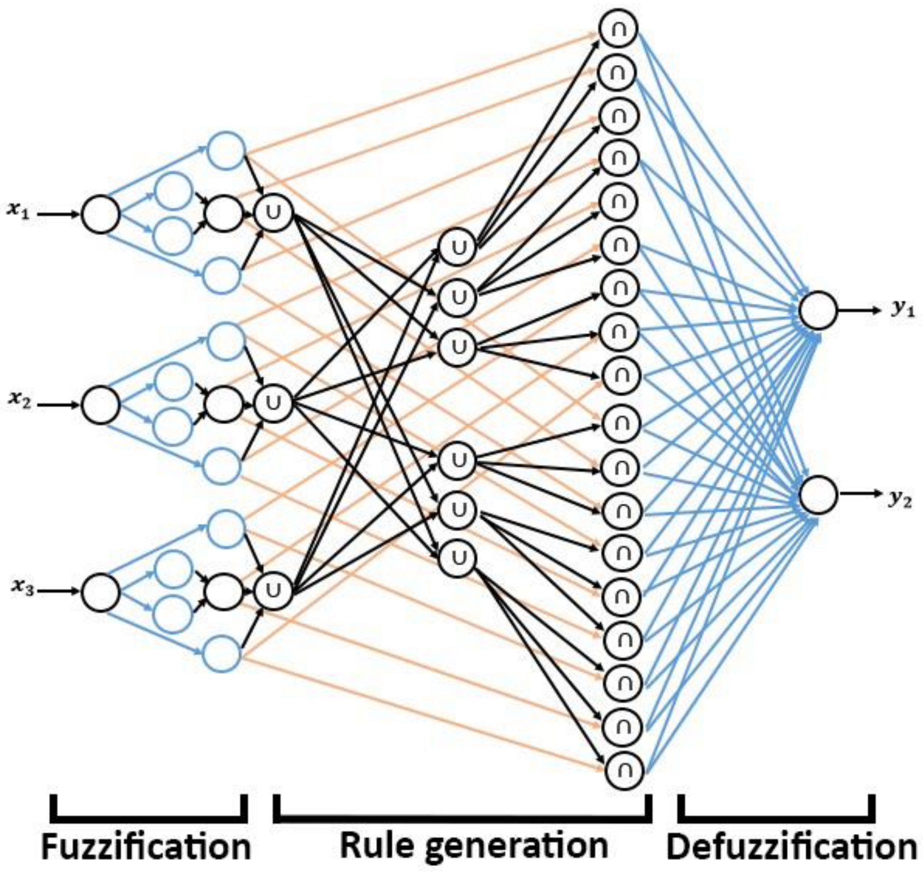


Figure 1. 15 : Fuzzy neural system architecture.

In figure 1.15, the blue connection lines are adjustable weights, the others are fixed weights. The empty black circles are neurons with sigmoid activation functions, and the other empty circles are neurons with linear activation functions. The neurons with (∪) and (∩) calculate the soft max and the soft min, respectively.

4.3. NEuro Fuzzy CONtrol (NEFCON)

NEFCON is a Neuro Fuzzy controller developed by Nauck et al. [138]. Its architecture is based on the generic fuzzy perceptron, which is an MLP with one hidden layer. The inputs of the NEFCON are the state variables of the controlled system and the outputs are the control signals. The neurons of the hidden layer represent the fuzzy rules. The weights in the NEFCON architecture are fuzzy subsets instead of real numbers. Therefore, the weights between the input layer and the hidden layer are the fuzzy subsets associated to the inputs, and the weights between the hidden layer and the output layer are the fuzzy subsets associated to the outputs (figure 1.16). The NEFCON algorithm is based on the Mamdani type fuzzy system.

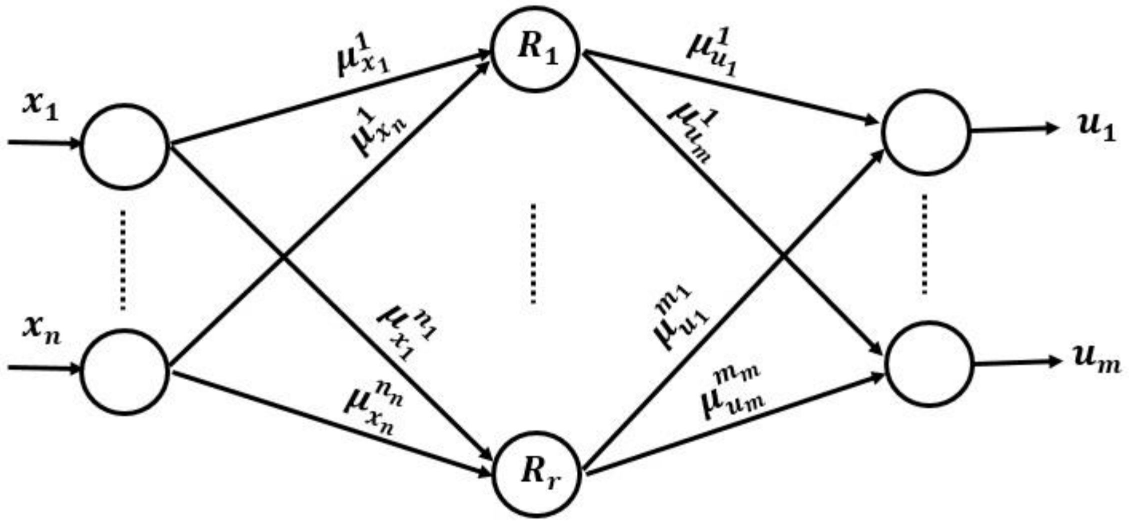


Figure 1. 16 : Neuro fuzzy Control architecture.

The NEFCON learning algorithm is based on the reinforcement learning, which does not require any supervision. Due to the nature of the network weights (fuzzy subsets), the learning algorithm uses a fuzzy error instead of crisp error in order to learn and optimize the fuzzy rule base. The NEFCON learning algorithm is detailed in [144].

5. <u>AI-based Nonlinear Model Predictive Control (AI-NMPC)</u>

AI-based NMPC algorithms are nonlinear model predictive controllers were at least one AI tool was used to design the control algorithm. Three major approaches of AI-NMPC can be distinguished and are given as follows:

- NMPC based on AI prediction models: in this approach, the used model to predict the future behavior of the controlled system is based on an AI method. Several AI- based prediction models can be found in the literature, such as:
  - ➢ Neural Network Based NMPC (NNMPC) [58,99–103,145,146].
  - ➢ Fuzzy model Based NMPC (FMPC) [96,97,147].
  - ➢ Fuzzy Neural Network based NMPC (FNNMPC) [148–150].
- NMPC based on AI optimization algorithms to solve the optimization problem: in this approach, a meta-heuristic algorithm is used to solve the optimization problem of the NMPC. Several NMPC using meta-heuristic algorithms exists, some of them are given as follows:
  - ➢ NMPC based on Genetic algorithm [104,105].
  - ➢ NMPC based on Particle Swarm Optimization algorithm [90,106,107].
  - ➢ NMPC based on Artificial Bee Colony [96,97].
  - ➢ NMPC based on Evolutionary Algorithm [108].
- NMPC based on AI optimization algorithms to determine the optimal values of the design parameters of the predictive controller: Some variants of this strategy are:

- NMPC using Imperialist Competitive algorithm to find optimal control parameters [151].
- NMPC based on sequential parameter optimization to find optimal control parameters [152].

Other AI-NMPC strategies can be found in the literature, which are generally hybrid versions of the above-mentioned strategies.

6. Conclusion

In this chapter, four AI-based control techniques were introduced; namely the neural network control strategy, the fuzzy logic control technique, the adaptive neuro-fuzzy inference system and the AI-based nonlinear model predictive control.

The use of artificial intelligence tools has allowed developing powerful, efficient, robust and adaptive control algorithms. These algorithms can be used to control a large class of nonlinear systems. The development of powerful processors allows the implementation of these algorithms, even complex ones, for real-time control applications.

CHAPTER 2

META-HEURISTIC OPTIMIZATION ALGORITHMS

1. Introduction

In this chapter, the fundamental concepts of meta-heuristic optimization algorithms and the formulation of the corresponding optimization problem are introduced. Some well-known meta-heuristic algorithms, namely the Particle Swarm Optimization (PSO), the Teaching Learning Based Optimization (TLBO) and some of their variants, are given. A proposed version of the TLBO algorithm, called the Enhanced TLBO (ETLBO), is presented in the last section of this chapter.

2. Basics of meta-heuristic optimization algorithms

Find an optimal solution for optimization problems is often a complex and a time-consuming task. Since, it is not possible to find an analytical solution to all optimization problems; several numerical methods have been developed. Besides most of these methods require the calculation of the cost function derivative, they suffer from the problem of convergence towards local minima. In fact, meta-heuristic algorithms have been proposed to deal with these problems and find good solutions at a reasonable computational cost. The concept of meta-heuristic has been introduced in 1945 [153] and has known, since then, wide dissemination among the research community. This interest has led to the emergence of several meta-heuristic algorithms that try to solve complex optimization problems.

The composed term “meta-heuristic”, in which the suffix “meta” is a Greek word that means “upper level methodology”, has been introduced by Glover in 1986 [154]. In mathematical optimization and computer science fields, meta-heuristic refers to a more efficient, higher level and general purpose optimization algorithms that may give a good solution to an optimization problem, especially with incomplete information or limited computation capacity. Meta-heuristic algorithms are iterative generation processes, which explore and exploit the search space to find efficient near-optimal solutions using learning strategies. They are based on the classical heuristic algorithms, the biological evolutions, the neural systems, and the statistical process [155].

2.1. Classification

Meta-heuristic algorithms can be classified according to several criteria:

2.1.1. Local search and global search

Meta-heuristic algorithms can be classified according to the used search strategy. Local search methods, such as the simulated annealing algorithm [156], the Tabu-search strategy [154], the variable neighborhood search algorithm [157] and the Greedy Randomized Adaptive Search Procedure (GRASP) [158] are used to find the local optimum. The Second class of meta-heuristic algorithms uses global searching methods, such as: Ant Colony Optimization (ACO) algorithm [159], particle swarm optimization (PSO) algorithm [160], Artificial Bee Colony (ABC) algorithm [161] and teaching learning based optimization algorithm [109,110].

2.1.2. <u>Single solution and population based</u>

Another classification is based on the number of candidate solutions. In fact, there are some algorithms that use a single candidate solution and other algorithms that use population of candidate solutions, to solve the optimization problem. Among the algorithms of the first class we can mention: the simulated annealing and the variable neighborhood search. These types of algorithms allow a deep search of local regions compared to algorithms based on population search. The population based search algorithms aim to solve the optimization problem using a group of candidate solutions called population. As examples of population based algorithms we can mention: the genetic algorithm, the ant colony optimization, the particle swarm optimization, the artificial bee colony, the teaching learning based optimization, etc. The single candidate solution based algorithms allow a deep search of local regions while the population based algorithms allow more efficient exploration of the search space.

2.1.3. <u>Memory usage and memory less algorithms</u>

This classification is based on the state of existence of memory units. Some meta-heuristic algorithms do not require information that is already collected, on the search space (for example the simulated annealing algorithm). However, other meta-heuristic algorithms require the previous extracted information to generate a new one, such as the PSO algorithm.

2.1.4. <u>Hybridization and mimetic algorithms</u>

Hybrid meta-heuristic algorithms combine a meta-heuristic method with other optimization techniques. Among these algorithms we find: the mathematical programming methods [162] and the constraint logic programming [163]. The components of a hybrid meta-heuristic algorithm can simultaneously operate and exchange information to solve the optimization problem. On the other hand, mimetic algorithms use only a meta-heuristic method to solve the optimization problem.

2.1.5. <u>Deterministic and stochastic algorithms</u>

Another classification is based on the state of existence of random operations. A stochastic meta-heuristic algorithm uses random parameters or distributions during the search. On the other hand, a deterministic meta-heuristic algorithm uses only deterministic decisions.

2.1.6. <u>Iterative and greedy algorithms</u>

Iterative meta-heuristic algorithms start with an initial solution and, using some search operators, iteratively adjust this solution until some criteria are verified. In greedy meta-heuristic algorithms, starting from an empty solution and assigning a single variable of the optimization problem in each step, the solution is built step by step.

2.1.7. <u>Nature-inspired and non-nature-inspired algorithms</u>

Another classification of meta-heuristic algorithms is based on whether they are inspired from the nature or not. Some of them, such as the ABC and the PSO are inspired from swarm intelligence; while others such as the simulated annealing is inspired from physics.

In this thesis, only stochastic iterative population-based meta-heuristic algorithms are used.

2.2. Population based meta-heuristic algorithms

Starting from an initial population of candidate solutions, the population based meta-heuristic algorithms iteratively manipulate the current population, using some search operators, to obtain better solutions. A part (or all) of the current population is replaced from the new generated population. This process of generation and replacement is continued until a stopping criterion is verified.

The main difference between all based meta-heuristic algorithms lies in the used generation/replacement strategy. Below are given the common steps of such algorithms.

Algorithm 2.1

```
Step 0: initialization
    • Set $i = 0$
    • Set the initial population ($X_{i=0}$)
Step 1: generation/replacement
    • For $i = 1$: maximum number of iterations
        o Generate the new population ($newX_i$)
        o Update some or all of the old population ($X_i$) using the new generated one ($newX_i$)
        o If the stopping criterion is satisfied, go to step 2.
    • End
Step2: output the best solution
```

2.2.1. Initial population

Regardless of the used meta-heuristic algorithm, a special attention must be given to the step of generating the initial population. An inappropriate choice of this population could greatly affect the efficiency of a given meta-heuristic algorithm; if the search space is not well covered, the optimization algorithm could converge towards a local optimum, or take a long time to find the appropriate solution. Despite the importance of this step in the implementation of any meta-heuristic optimization algorithm, there is only few published works that addresses the problem of how to generate a good initial population [164–166].

According to [167], the following four approaches to generate an initial population can be considered:

- Random generation
- Sequential diversification
- Parallel diversification
- Heuristic initialization

2.2.2. Population size

The population size is one of the most critical parameters of any population-based meta-heuristic algorithm; a large size could gives good solutions of the optimization problem, but at the expense of increasing the computational time, while a small one could generate poor solutions. In real-time applications, a balance has to be found between the computational time and the accuracy of solutions. There is no general strategy to determine the optimal size of the population according to the considered optimization problem. However, some particular strategies, which depend on the used meta-heuristic algorithm and the optimization problem dimension, have been proposed [167–169]. Instead of using a population with a fixed size, some works have suggested dynamically increasing and decreasing the population size during the optimization process [170]. In most applications, the population size is chosen according to the dimension of the optimization problem [171].

2.2.3. Exploration and exploitation

Population based meta-heuristic algorithms are characterized by their ability to explore the search space looking for regions of interest, and exploit these regions to find the optimum, or the near optimum, solution. These proprieties are somewhat exclusive and a tradeoff between the exploration and the exploitation should be established to have an efficient optimization algorithm. Meta-heuristic algorithms use several mechanisms to make this balance. If the exploration has not been thorough, one or more regions of interest could be missed and consequently the global optimum or even a solution in its vicinity cannot be found; the algorithm will be trapped in local optimum causing its premature convergence. The convergence speed could also be affected by an inappropriate exploration; the algorithm will take more time to search the regions of interest. On the other hand, if the exploitation process is prematurely stopped, the algorithm may miss a good, or even, an optimum solution. So, it is important to give sufficient time for the exploitation operation. It is obvious that an excessive exploitation slows down the convergence speed of the algorithm.

2.2.4. Stopping criteria

According to the nature of the optimization problem, several strategies to stop the optimization process can be used [167]. Some of them are given as follows:

- Static strategy: in this case, the end of the optimization process is known a priori. Several criteria for this strategy can be used, such as the maximum number of iterations and the fixed number of objective function evaluations. This strategy is used in real time applications, where the maximum time for the optimization process is limited.
- Adaptive strategy: in this case, the end of the optimization process cannot be known a priori. Several criteria for this strategy can be used, such as the maximum number of non-improving iterations and the error tolerance that indicates a satisfactory solution is reached.

3. Solving an optimization problem using meta-heuristic algorithms

3.1. Formulating of the optimization problem and basic algorithm

The standard form of any optimization (in this case minimization) problem is given as follows:

$$min_X \, F(X) \tag{2.1}$$

Subject to:

$$\begin{gathered} G(X) = 0 \\ H(X) \leq 0 \end{gathered} \tag{2.2}$$

where: $X \in \mathbb{R}^n$ are the variables to be optimized, n is the number of optimized variables, $F(X)$ is the cost function and $G(X)$ and $H(X)$ are the constraints functions.

Assuming that $S \subset \mathbb{R}^n$ is the subset of admissible solutions that satisfy the constraints given by equation (2.2) and n_p is the chosen population size. Using a population based meta-heuristic algorithm, the basic steps needed to solve the above optimization problem are summarized as follows:

Algorithm 2.2

```
Step 0: initialization
    • Set $i = 0$
    • For $j = 1: n_p$
        o Choose the initial solution ($X_0^j$) from $S$, such as $X_0^j \in S$.
    • End
Step 1: cost function evaluation
    • For $j = 1: n_p$
        o Using the candidate solution ($X_i^j$), evaluate the cost function given by equation (2.1).
    • End
Step 2: generation/replacement
    • For $j = 1: n_p$
        o Generate a new candidate solution ($newX_i^j$)
        o Apply an updating strategy.
        o If the replacement is necessary
            ➢ $X_i^j = newX_i^j$
        o End if
    • End
Step 3: finding the best solution
    • $X_i^{best} = X_i^1$
    • For $j = 2: n_p$
        o If $F(X_i^j)<F(X_i^{best})$
            ➢ $X_i^{best} = X_i^j$
        o End if
    • End
Step 4:
    • If stopping criteria is satisfied
        o Report $X_i^{best}$ as the solution
        o Exit the optimization process
    • Else
```

  - $i = i + 1$
  - Go to step 1.
- End if

As each algorithm has its own particularities, algorithm 2.2 does not accurately describe all existing population based algorithms.

3.2. Constraints handling

In order to solve the optimization problem given by equations (2.1) and (2.2), algorithm 2.2 has been given under the assumption that the subset S of all admissible solutions is known a priori. However, it is very hard and sometimes impossible to create the subset S. Indeed, even if this subset is known a priori, several difficulties could appear while generating the new candidate solutions. There are several approaches that can be used to handle constraints, they can be grouped into the following categories [167]:

- Reject strategy: it also called the death penalty approach. It is based on the rejection of all unfeasible candidate solutions (solutions that do not satisfy the constraints given by equation (2.2)). This approach can only be used if the majority of the search space is feasible. The unfeasible solutions are not used to gather information about global solutions that can be either on the boundary between feasible and infeasible solutions, or on another independent feasible region, if the admissible set contains discontinuous regions. This approach is unsuccessful with most of the optimization problems.
- Penalizing strategy: this approach is the most popular and used method to handle the imposed constraints. In this method, both feasible and unfeasible solutions could be considered, and the original cost function is modified to include new terms that will heavily penalize infeasible solutions.
- Repairing strategy: this approach deals with unfeasible solutions by transforming them into feasible solutions using custom build heuristic algorithms. Therefore, the optimization performance is highly depending on the efficiency of the custom algorithms. Due to the added step of repairing unfeasible solutions, this approach takes more computing time than other approaches, making it impractical for fast real-time applications.
- Preserving strategy: in this approach, using some specific knowledge about the handled problem, the optimization algorithm is modified to ensure that all generated candidate solutions are feasible. Clearly, the modified optimization algorithm cannot be used for a different optimization problem. In addition, the initial solutions must also be feasible.

4. Variants of meta-heuristic algorithms

In the aim of designing efficient optimization algorithms, several meta-heuristic optimizers have been developed and compared. It has observed that each meta-heuristic optimizer outperforms the others on some other cost functions. Based on the “no free lunch theorem”, Wolpert and Macready have stated that all meta-heuristic algorithms perform exactly the same, according to any performance measure, when averaged over all possible cost functions [172,173]. However, in practical situations, the number of interesting cost functions is quite small and the goal is to determine the best optimization meta-heuristic algorithm against these functions.

4.1. Particle swarm optimization algorithm

4.1.1. Original PSO algorithm

Particle Swarm Optimization is an iterative, stochastic, population-based meta-heuristic algorithm that have developed by Kennedy J. and Eberhart R. [160,174] to solve a continuous single-objective problem. PSO mimics the behavior of swarms when collaboratively searching for food sources, such as: swarm of insects, flocks of birds, herds of animals and schools of fish. Each member of the swarm, called particle, tries to find the best food source by learning from his or her personal experience and the global swarms' experience. This phenomenon has been mathematically modeled in the form of an optimization algorithm.

In the PSO algorithm, a randomly distributed population (called a swarm) of candidate solutions (called particles) is generated. These particles are moved around in the search space and their suitability is evaluated using an appropriate fitness function. The best position, founded by the whole swarm, represents the global optimum of the considered optimization problem. The particles change their positions in the search space to improve their fitness' values and provide more accurate solutions. This change is based on the best position of each particle in the search space, the entire swarm's best position and some random behavior. The discovered improved positions will guide the swarm's movements in the next iteration. This process is iteratively repeated until a satisfactory solution is found.

The first step of the PSO algorithm is to generate a random initial population using the following equation:

$$X_i^j = rand_i^j\left(X_{max}^j - X_{min}^j\right) + X_{min}^j \tag{2.3}$$

such as $i = [1,2,\dots,n_p]$ and $j = [1,2,\dots,D]$.

The function $rand_i^j$ generates a random number in the range [0, 1], D is the dimension of the optimization problem, $n_p$is the number of particles (the population size), and X_{max}^j and X_{min}^j are the upper limit and the lower limit of the j^{th} dimension' search space, respectively.

The second step is to evaluate the fitness of each particle ($F(X_i)$), where $X_i = [X_i^1, X_i^2, \dots, X_i^D]$, determine the personal best solution of every particle ($P_i = [P_i^1, P_i^2, \dots, P_i^D]$) and find the global best solution ($G = [G^1, G^2, \dots, G^D]$). This step is performed for all particles in the swarm ($i = 1,2,\dots,n_p$) and the entire dimension of the optimization problem ($j = 1,2,\dots,D$).

The third step is to update the velocity ($V_i = [V_i^1, V_i^2, \dots, V_i^D]$) and the position ($X_i = [x_i^1, x_i^2, \dots, x_i^D]$) of each particle ($i = 1,2,\dots,n_p$) in the entire dimension of the search space ($j = 1,2,\dots,D$), according to the following equations:

$$V_i^j(k+1) = V_i^j(k) + c_1 rand1_i^j(P_i^j - X_i^j) + c_2 rand2_i^j(G^j - X_i^j) \tag{2.4}$$

$$X_i^j(k+1) = X_i^j(k) + V_i^j(k+1) \tag{2.5}$$

Where:

$rand1_i^j$, $rand2_i^j$ are random numbers in the range [0, 1], and c_1, c_2 are acceleration constants. It is up to the user to determine these parameters according to the handled optimization problem.

The complete steps of the PSO algorithm are summarized by the flowchart given in figure 2.1.

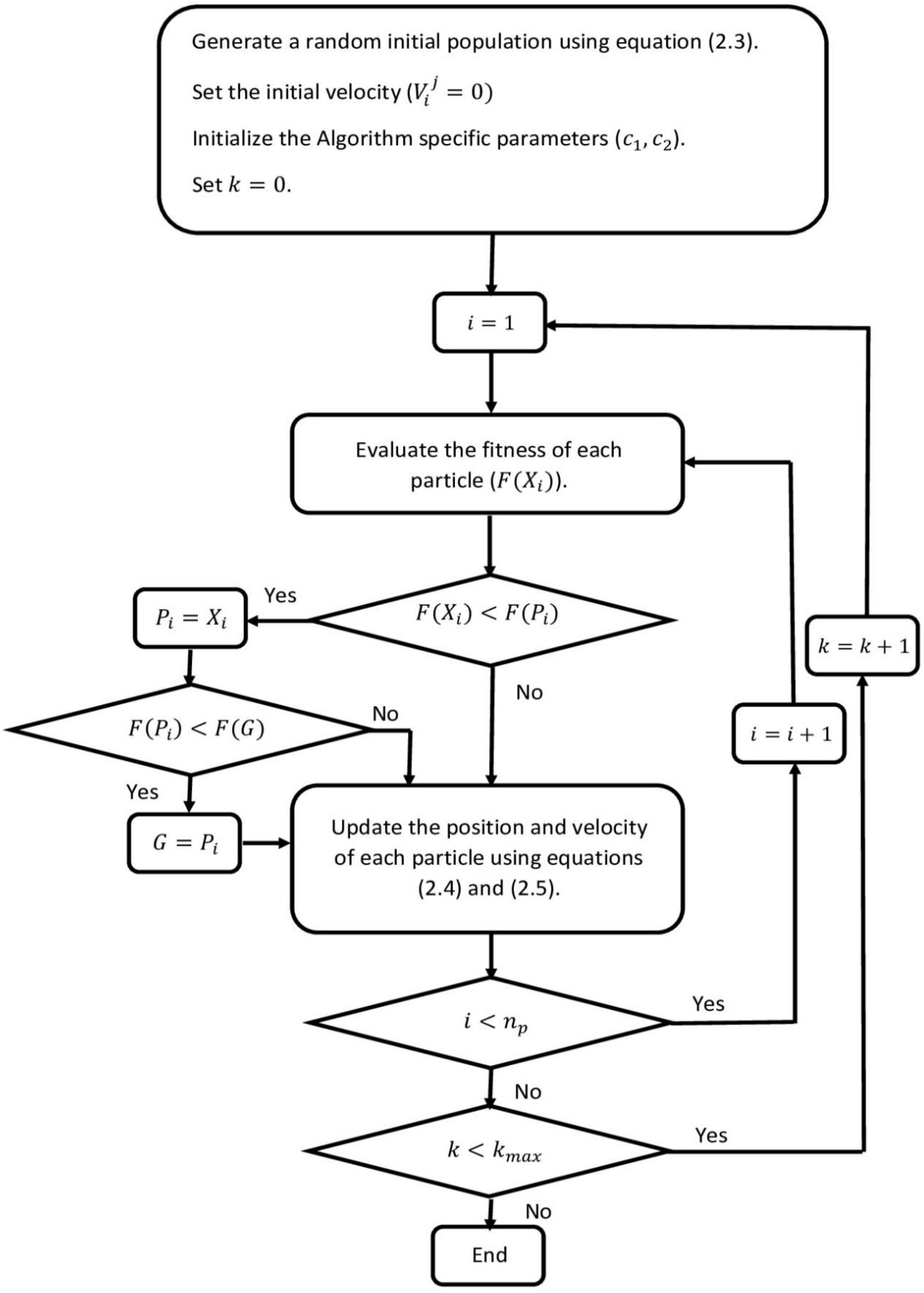

Generate a random initial population using equation (2.3).
Set the initial velocity ($V_i^j = 0$)
Initialize the Algorithm specific parameters (c_1, c_2).
Set $k = 0$.
$i = 1$
Evaluate the fitness of each particle ($F(X_i)$).
$F(X_i) < F(P_i)$
Yes
$P_i = X_i$
No
$F(P_i) < F(G)$
No
Yes
$G = P_i$
Update the position and velocity of each particle using equations (2.4) and (2.5).
$i < n_p$
Yes
$i = i + 1$
No
$k < k_{max}$
Yes
$k = k + 1$
No
End

Figure 2. 1 : Flow chart of the PSO algorithm.

4.1.2. Some variants of the PSO algorithm

The original PSO algorithm has a problem of balance between the exploration of the search space and the exploitation of prominent regions [175,176]. It can be trapped in local optimums, especially in the case of multimodal, rugged, and non-separable optimization problems. To deal with this problem, several variants of the PSO algorithm have been proposed. The main changes in three variants of the PSO algorithm are summarized as follows:

- A modified particle swarm optimizer using a fixed inertia weight [177]: in this version, a new parameter, called inertia weight, has been added to the velocity updating equation of the original PSO algorithm. This equation becomes:

$$V_i^j(k+1) = \omega \cdot V_i^j(k) + c_1 rand1_i^j(P_i^j - X_i^j) + c_2 rand2_i^j(G^j - X_i^j) \quad (2.6)$$

  Exploration and exploitation abilities have been balanced using the inertia weight (ω), where, a small value of ω gives more importance to exploitation than to exploration (local search) and a large number of ω gives more importance to exploration than to exploitation (global search).
- Adaptive particle swarm optimization [177,178]: this method has the same velocity updating equation as the modified particle swarm optimizer using a fixed inertia weight. However, instead of using the fixed inertia weight during the optimization process, a dynamic ω is used. Several methods for changing the value of ω have been proposed. The most common is the linearly decreasing technique [177], it is given by the following equation:

$$\omega(k+1) = \omega_d \cdot \omega(k) \quad (2.7)$$

  where: ω_d is a constant belongs to $[0, 1]$.
  Other methods, such as the fuzzy adaptive particle swarm optimization [178], can be found in the literature.
- Comprehensive learning particle swarm optimizer [179]: this strategy is based on the comprehensive learning strategy. The velocity updating equation is given as follows:

$$V_i^j(k+1) = \omega \cdot V_i^j(k) + c \cdot rand_i^j(P_{f_i(j)}^j - X_i^j) \quad (2.8)$$

  where: $f_i = [f_i(1), f_i(2), \dots, f_i(D)]$ defines which particle' personal best (P_i^j) that the particle (X_i^j) should follow.
  In the original PSO algorithm, each particle follows its own personal best and the global best. However, in this variant, each particle can follow the personal best of any particle of the swarm, including its own. This process is randomly performed according to the following equation:

$$f_i(j) = rand_i^j \quad (2.9)$$

  If $f_i(j) > Pc_i^j$: the particle (X_i^j) will learn from its own personal best (P_i^j).
  If $f_i(j) < Pc_i^j$: the particle (X_i^j) will learn from another particle' personal best (P_i^j) using the tournament selection procedure as explained in [179].
  $rand_i^j$ is a random number in the range [0 1], Pc is the learning probability, which can take different values for different particles.

4.2. Teaching Learning Based Optimization algorithm

The TLBO is a teaching-learning process inspired algorithm that was proposed by Rao et al. [109,110]. TLBO is a meta-heuristic algorithm that uses a group of individuals (population) to search for the global solution. Its population is considered as a group of students. The TLBO working principle is divided into two phases, teacher and learners phases. In the teacher phase, students learn from their teacher, the teacher gives the best performance in the population. In the learners phase, students learn by randomly interacting between themselves. The cost function variables are the taught subjects, and the fitness value of the optimization problem is considered as the students' results.

4.2.1. Teacher phase

The TLBO algorithm begins with the teacher phase where students learn from their teacher. Assuming that the optimization problem is formulated as follows:
m variables (subjects to be taught) exist with n students (population size), at any iteration i, this phase is decomposed into the following steps:

- A teacher is selected from the population by choosing the student who gives the best fitness $X^i_{j\,k_{best}} (j = 1, \dots, m,\ k = 1, \dots, n)$.
- Calculate the students mean result M^i_j and the difference mean $d^i_{j\,k}$ for each subject as follows:

$$M^i_j = \frac{\sum_{k=1}^{n} X^i_{j\,k}}{n} \tag{2.10}$$

$$d^i_{j\,k} = r(X^i_{j\,k_{best}} - T_F M^i_j) \tag{2.11}$$

where r is a random number between 0 and 1, the Teaching Factor T_F is a random integer number between 1 and 2.

- Updating each existing solution as follows:

$$Xnew^i_{j\,k} = X^i_{j\,k} + d^i_{j\,k} \tag{2.12}$$

- A greedy selection is applied between the old and the new solution ($X^i_{j\,k}$, $Xnew^i_{j\,k}$) in order to keep the best solution.

4.2.2. Learners phase

In this phase, students try to improve themselves by interacting with each other. The student X_A choose randomly to interact with the student X_B and learn from him. This phase is decomposed into the following steps:

- Choose randomly q pairs of solutions such that : $F^i_A \neq F^i_B$, where F^i_A and F^i_B are the fitness values of X_A and X_B , respectively.
- In case of minimization problem, and for each pair, update the solutions using the following equations:

$$Xnew^i_{j\,A} = X^i_{j\,A} + r\left(X^i_{j\,A} - X^i_{j\,B}\right), \quad if \quad F^i_A < F^i_B \tag{2.13}$$

$$Xnew^i_{j\,A} = X^i_{j\,A} + r\left(X^i_{j\,B} - X^i_{j\,A}\right), \quad if \quad F^i_B < F^i_A \tag{2.14}$$

- A greedy selection is applied between the old and the new solution to keep the best solution.

4.2.3. Initialization step

Before starting the TLBO algorithm, the following parameters must be initialized:

- the dimension "m" of the optimization problem (m = number of optimized variables).
- the population size "n". The choice of the population size has a large impact on the meta-heuristic algorithms performance. So choosing the population size depend on several factors, two of them are the converging speed and the solution accuracy [180,181].
- the maximum number of iterations k_{max}. The choice of the value of this parameter depends on the chosen sampling time.
- the admissible maximum and minimum values for each variable $X_{j_{min}}$ and $X_{j_{max}}$.
- the initial solutions $X_{j\,k}^{1}$ are randomly chosen using the following equation:

$$X_{j\,k}^{1} = rand(0,1) * \left(X_{j_{max}} - X_{j_{min}} \right) + X_{j_{min}} \qquad (2.15)$$

- the termination criterion ε.

To illustrate the different steps of the TLBO method, its flow chart is given in figure (2.2).

4.3. Improved teaching-learning-based optimization algorithm

In order to enhance the optimization performance of the original TLBO algorithm, Rao and Patel proposed an Improved TLBO (I-TLBO) to solve unconstrained optimization problems [182]. In this variant, the exploration and the exploitation capabilities were enhanced by introducing the concept of multi-teachers, an adaptive teaching factor, tutorial training and self-motivated learning. Using several unconstrained benchmark functions, it has been proven that the improved TLBO algorithm gives better optimization performance than the original TLBO, the ABC algorithm, the modified ABC algorithm, several versions of the PSO and other optimization algorithms [182].

The modifications to the basic TLBO algorithm are given as follows:

- Number of teachers: in the original TLBO algorithm, only one teacher is chosen to teach the entire population. However, in the I-TLBO algorithm, several teachers are selected. Assuming that the chosen number of teachers is equal to "T_n", the best candidate solution is selected as the first teacher $(X_{teacher})_1$ such as $F_{(X_{teacher})_1} = F_{best}$, where $F_{(X_{teacher})_1}$ is the first teacher' fitness value and F_{best} is the fitness value of the best candidate solution. The remaining $(T_n - 1)$ teachers are selected as follows:

$$F_{(X_{teacher})_s} = F_{(X_{teacher})_1} - rand^{*} \qquad (2.16)$$

where: $s = 2,3, \dots, T_n$ and $rand^{*}$ is a random number in the range $[0, (F_{max} - F_{min})]$. F_{max} and F_{min} are the maximum fitness and the minimum fitness of all learners, respectively.

If the equality is not satisfied, select $(X_{teacher})_s$ as the element that gives the closest value to $F_{(X_{teacher})_s}$ calculated above. After that, assign a group of learners to each teacher as follows:

For $k = 1: (n - T_n)$

- If $F_{(X_{teacher})_1} \geq F_{(X)_k} > F_{(X_{teacher})_2}$
  - Assign the learner $(X)_k$ to the teacher $(X_{teacher})_1$
- Else, If $F_{(X_{teacher})_2} \geq F_{(X)_k} > F_{(X_{teacher})_3}$
  - Assign the learner $(X)_k$ to the teacher $(X_{teacher})_2$

$\vdots$

- Else, If $F_{(X_{teacher})_{T_n-1}} \geq F_{(X)_k} > F_{(X_{teacher})_{T_n}}$
  - Assign the learner $(X)_k$ to the teacher $(X_{teacher})_{T_n-1}$
- Else
  - Assign the learner $(X)_k$ to the teacher $(X_{teacher})_{T_n}$

End

- Adaptive teaching factor (T_F): the second change is related to the teaching factor. Such as, in the basic TLBO algorithm the teaching factor is chosen randomly and it can be either one or two. Although, in the I-TLBO algorithm, for each teacher, the teaching factor is calculated as follows

  If $F_{(X_{teacher})_k} \neq 0$

$$(T_F)_k = \frac{F_{(X)_{rand}}}{F_{(X_{teacher})_k}} \qquad (2.17)$$

  where: $k = 1{:}T_n$, $F_{(X_{teacher})_k}$ is the fitness value of the k^{th} teacher, and $F_{(X)_{rand}}$: is the fitness value of a random learner who is assigned to the k^{th} teacher.
- Learning through tutorial: the third modification is based on the fact that, during tutorial hours, learners can learn by interacting with themselves or with their teachers. Therefore, equation (2.12) becomes:

$$(Xnew^i_{j\,k})_s = (X^i_{j\,k} + d^i_{j\,k})_s + rand \cdot \left(X^i_{j\,hh} - X^i_{j\,k}\right), \qquad if\ F_{(X)_{hh}} > F_{(X)_k} \qquad (2.18)$$

$$(Xnew^i_{j\,k})_s = (X^i_{j\,k} + d^i_{j\,k})_s - rand \cdot \left(X^i_{j\,hh} - X^i_{j\,k}\right), \qquad if\ F_{(X)_{hh}} < F_{(X)_k} \qquad (2.19)$$

  where:

  i: is the current iteration, $j = 1, \dots, m,\ k = 1, \dots, n$, m is the number of variables, n is the population size, $s = 1, \dots, T_n$, and $X^i_{j\,hh}$ is a random learner.

- Self-motivated learning: in the original TLBO algorithm, learners can improve their grades (fitness) by learning from their teacher or by interacting with themselves. In the I-TLBO algorithm, the student can also improve their grades by self-learning. Hence, equations (2.13) and (2.14) become:

$$Xnew^i_{j\,A} = X^i_{j\,A} + rand\left(X^i_{j\,A} - X^i_{j\,B}\right) + rand((X_{teacher})_1 - E_F X^i_{j\,A}), \quad if \quad F^i_A < F^i_B \qquad (2.20)$$

$$Xnew^i_{j\,A} = X^i_{j\,A} + rand\left(X^i_{j\,B} - X^i_{j\,A}\right) + rand((X_{teacher})_1 - E_F X^i_{j\,A}), \quad if \quad F^i_B < F^i_A \qquad (2.21)$$

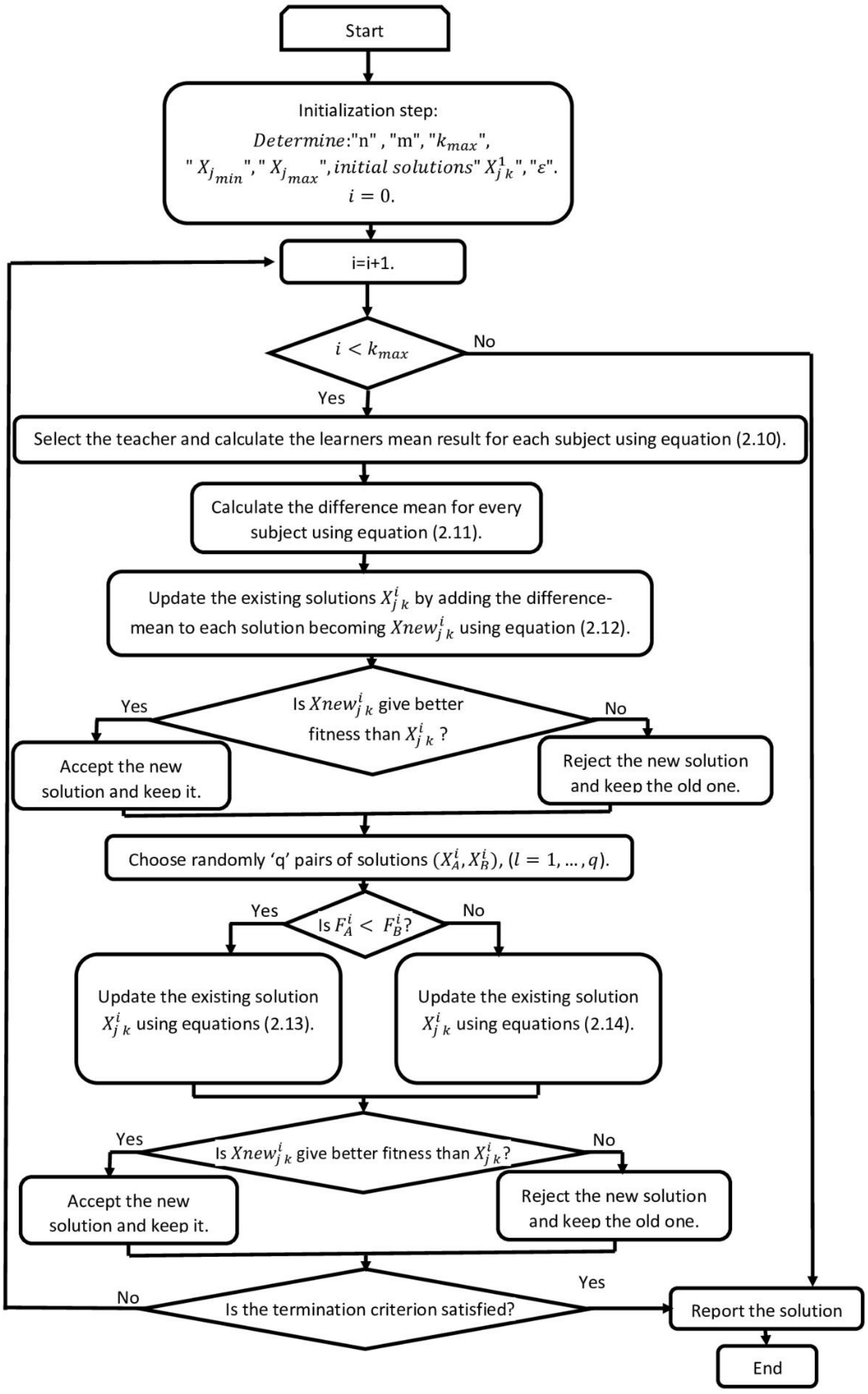

Start
Initialization step:
$Determine$:"n" , "m", "k_{max}",
" $X_{j_{min}}$"," $X_{j_{max}}$", $initial\ solutions$" $X^1_{j\,k}$", "ε".
$i = 0$.
i=i+1.
$i < k_{max}$
No
Yes
Select the teacher and calculate the learners mean result for each subject using equation (2.10).
Calculate the difference mean for every subject using equation (2.11).
Update the existing solutions $X^i_{j\,k}$ by adding the difference-mean to each solution becoming $Xnew^i_{j\,k}$ using equation (2.12).
Is $Xnew^i_{j\,k}$ give better fitness than $X^i_{j\,k}$?
Yes
No
Accept the new solution and keep it.
Reject the new solution and keep the old one.
Choose randomly 'q' pairs of solutions (X^i_A, X^i_B), $(l = 1, \ldots, q)$.
Is $F^i_A < F^i_B$?
Yes
No
Update the existing solution $X^i_{j\,k}$ using equations (2.13).
Update the existing solution $X^i_{j\,k}$ using equations (2.14).
Is $Xnew^i_{j\,k}$ give better fitness than $X^i_{j\,k}$?
Yes
No
Accept the new solution and keep it.
Reject the new solution and keep the old one.
Is the termination criterion satisfied?
No
Yes
Report the solution
End

Figure 2. 2 : Flow chart of the TLBO algorithm.

4.4. The proposed Enhanced Teaching Learning Based Optimization (ETLBO)

In order to improve the performance of the TLBO algorithm, mainly the convergence rate, the selecting process of the students' pairs of the original algorithm is modified as follows.

4.4.1. Students pairs selecting process

The learners' phase of the original TLBO algorithm is based on three steps: randomly selecting of q pairs of solutions, updating these solutions according to equations (2.13) and (2.14), and Appling a greedy selection to the new obtained solutions.

The choice of the students' pairs in the learners' phase of the ETLBO algorithm is a critical task. Instead of using a random approach to choose these pairs, a more selectable approach, based on each student grade during the optimization process, is applied. It would be more interesting if students interact with their pairs having a good grade. This idea can be performed by introducing a new factor, named student grade "SG", it is given by:

$$SG_k^i = 100\frac{(\max(F^i) - F_k^i)}{(\max(F^i) + \varepsilon)} \tag{2.22}$$

Such as:

$F^i = [F_1^i, F_2^i, \dots, F_k^i, \dots, F_n^i]$, $i = 1, \dots, k_{max}$, $k = 1, \dots, n$, k_{max} is the maximum number of iterations, n is the population size and ε is a positive small number.

The pairs of students are chosen according to the following steps:

- Rank the students, from best to worst, according to their SG_k^i values. The biggest SG_k^i value indicates the best student and the lowest indicates the worst student.
- Randomly choose a percentage ($I_F \in [30\%,\ 50\%]$) to split the population into two groups, good and bad students, according to theirs grades, such as:

$$G_S = \frac{I_F}{100} * n,\ \ B_s = \left(1 - \frac{I_F}{100}\right) * n \tag{2.23}$$

where:

G_S is the number of good students and B_s is the number of bad students.

- Choose q pairs of solutions X_A and X_B. X_A and X_B are randomly chosen from the whole population and the group of good students, respectively.

5. Experimental study using several benchmark functions

In this section, using eight benchmark functions with different dimensions and search spaces, the quality of the proposed ETLBO algorithm is investigated. The obtained results using the ETLBO algorithm are compared with the results of the original TLBO algorithm, the I-TLBO algorithm for ($T_n = 1$ and $T_n = 2$), and the modified PSO algorithm (w-PSO). Using multiple tests, the parameters of w-PSO are chosen as follow:

$\omega(0) = 1,\ \ \omega_d = 0.99,\ \ V_{max} = 10,\ \ c_1 = 2,\ \ c_2 = 2$, and $|V_i^d| = V_{max}$ for $|V_i^d| > V_{max}$.

5.1. Benchmark functions description

The following eight well-known benchmark functions are used:

- Sphere function:

$$f_1(X) = \sum_{i=1}^{D} x_i^2 \ , \qquad -100 \le x_i \le 100 \tag{2.24}$$

$f_1(X)$ is a separable and multimodal function.

- Rosenbrock function:

$$f_2(X) = \sum_{i=1}^{D-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \ , \quad -2.048 \le x_i \le 2.048 \tag{2.25}$$

$f_2(X)$ is a non-separable and unimodal function

- Ackley function:

$$f_3(X) = 20 + \exp(1) - 20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D} cos(2\pi x_i)\right) \tag{2.26}$$

$$such\ as -32.768 \le x_i \le 32.768$$

$f_3(X)$ is a non-separable and multimodal benchmark function.

- Griewank function:

$$f_4(X) = 1 + \frac{1}{4000}\left(\sum_{i=1}^{D}(x_i - 100)^2\right) - \left(\prod_{i=1}^{D}\cos\left(\frac{x_i - 100}{\sqrt{i}}\right)\right) \tag{2.27}$$

$$such\ as -600 \le x_i \le 600$$

$f_4(X)$ is a non-separable and multimodal function.

- Weierstrass function:

$$f_5(X) = \sum_{i=1}^{D}\left(\sum_{k=0}^{20}\left[0.5^k\cos\left(2\pi 3^k(x_i + 0.5)\right)\right]\right) - D\sum_{k=0}^{20}[0.5^k\cos(3^k\pi)] \tag{2.28}$$

$$such\ as \quad -0.5 \le x_i \le 0.5$$

$f_5(X)$ is a separable and multimodal function.

- Rastrigin function:

$$f_6(X) = \sum_{i=1}^{D}(x_i^2 - 10\cos(2\pi x_i) + 10), \qquad -5.12 \le x_i \le 5.12 \tag{2.29}$$

$f_6(X)$ is a separable and multimodal benchmark function.

- NCrastrigin function:

$$f_7(X) = \sum_{i=1}^{D}(y_i^2 - 10\cos(2\pi y_i) + 10),$$

$$y_i = \begin{cases} x_i, & if \quad |x_i| < 0.5 \\ \dfrac{round(2x_i)}{2}, & if \quad |x_i| \ge 0.5 \end{cases} \quad such\ as -5.12 \le x_i \le 5.12 \tag{2.30}$$

$f_7(X)$ is a separable and multimodal benchmark function.

- Schwefel function:

$$f_8(X) = -\sum_{i=1}^{D}\left(x_i\sin\left(\sqrt{|x_i|}\right)\right) \ , \qquad -500 \le x_i \le 500 \tag{2.31}$$

Where:

$X = [x_1, x_2, \dots, x_D]$, D denote the dimension of all benchmark functions mentioned above.

$f_8(X)$is a separable and multimodal function.

The global minimum of all these benchmark functions is equal to zero at $X = [0,0, \dots, 0]$, except in the case of the Rosenbrock function $X = [1,1, \dots, 1]$.

5.2. Numerical Results

For all considered algorithms the population size is equal to 40 and the maximum number of iterations k_{max} is equal to 1000. Using the same software and hardware configuration, the optimization process of all considered benchmark functions is repeated 500 times. If the value of the evaluated function drops below 2.22e-16, it is reported as 0.

The obtained results are presented in table 2.1, where Mean and SD denote the mean value and the standard deviation of each benchmark function. In each row, the minimum mean value is indicated using a bold font. If the obtained mean value is equal to zero for more than one algorithm, the reported best result is that of the algorithm which requires less iterations to achieve the objective.

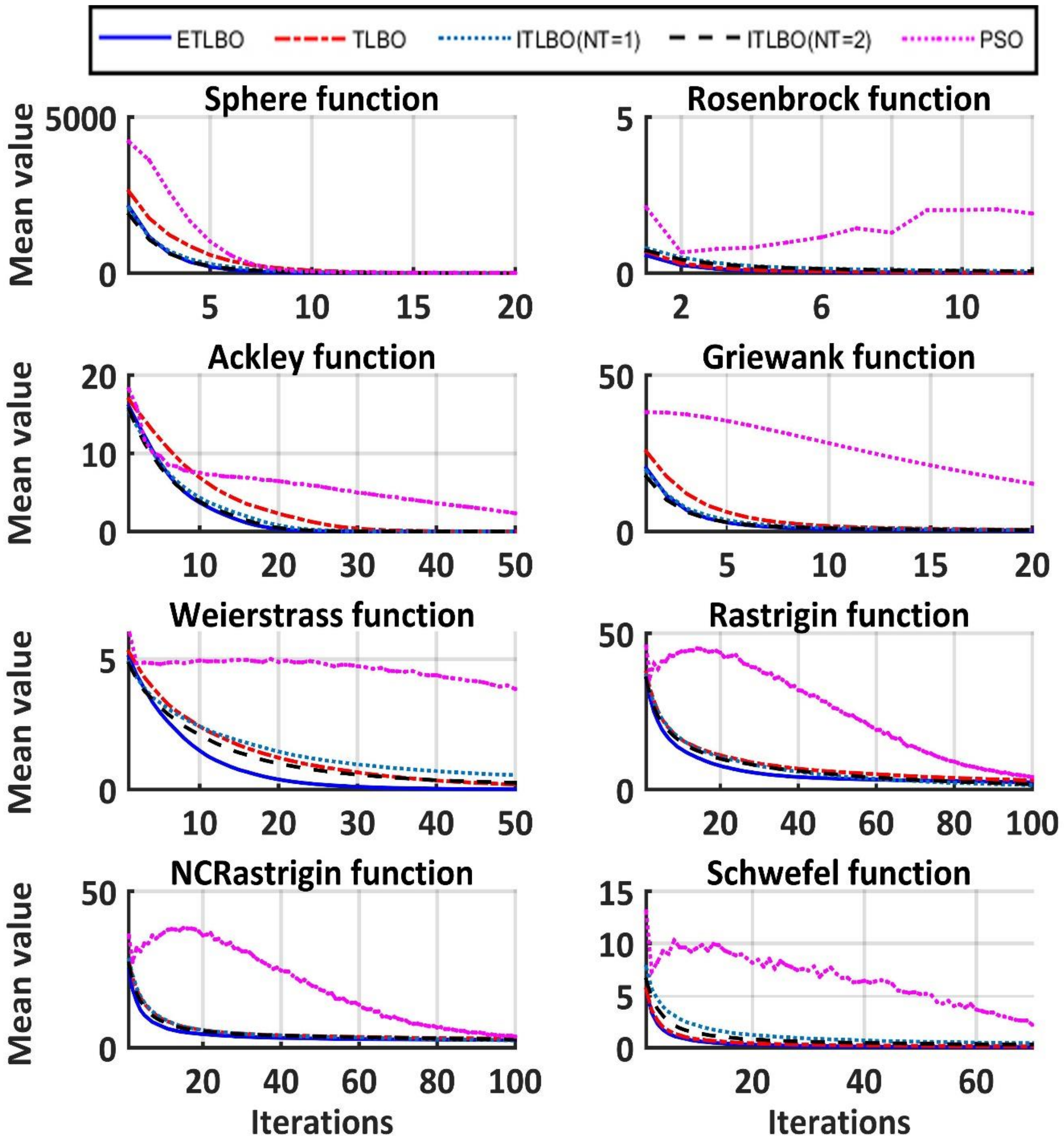


Figure 2. 3 : Convergence speed (D=2 for Rosenbrock function and D=5 for other functions).

	D	ETLBO		TLBO		I-TLBO ($T_n = 1$)		I-TLBO ($T_n = 2$)		w-PSO	
		Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
f_1	5	**0**	0	0	0	0	0	**0**	0	0	0
	10	**0**	0	0	0	0	0	0	0	1.22e-14	2.2e-13
	25	**0**	0	0	0	0	0	0	0	8.76e-4	3.29e-3
f_2	2	**0**	0	0	0	0	0	0	0	0	0
	3	**1.89e-9**	6.15e-9	5.85e-9	6.06e-8	3.12e-7	2.26e-6	2.90e-7	2.75e-6	2.67e-3	3.09e-3
	10	6.81e0	0.35e0	3.016e0	1.129e0	2.86e0	7.76e-1	**2.60e0**	7.14e-1	4.52e0	1.51e0
f_3	5	1.71e-15	1.49e-15	9.02e-16	2.24e-16	8.88e-16	0	**8.88e-16**	0	2.20e-15	1.72e-15
	10	4.17e-15	9.65e-16	4.31e-15	6.62e-16	**2.45e-15**	1.76e-15	2.57e-15	1.77e-15	3.29e-3	7.36e-2
	25	**4.43e-15**	1.59e-16	4.45e-15	1.59e-16	4.44e-15	0	4.44e-15	4.44e-15	3.05e-1	5.65e-1
f_4	5	**2.70e-4**	2.1e-3	1.96e-2	1.19e-2	1.41e-2	2.03e-2	1.69e-2	2.13e-2	1.08e-1	8.52e-2
	10	**1.70e-4**	1.8e-3	1.21e-2	1.62e-2	4.91e-3	1.66e-2	8.97e-3	2.81e-2	1.98e-1	1.53e-1
	25	**0**	0	1.38e-4	2.13e-3	0	0	0	0	1.15e0	6.31e-1
f_5	5	6.16e-4	7e-3	**1.30e-13**	2.90e-12	2.67e-2	1.23e-1	4.58e-3	3.01e-2	2.42e-3	1.95e-2
	10	**2.00e-4**	3.3e-3	5.65e-4	6.09e-3	1.74e-3	1.94e-2	3.71e-3	3.70e-3	1.05e-1	1.96e-1
	25	**0**	0	0	0	0	0	0	0	2.69e0	1.56e0
f_6	5	6.52e-3	1.45e-1	7.28e-2	2.63e-1	9.95e-3	2.22e-1	**0**	0	5.51e-1	6.92e-1
	10	**0**	0	2.81e0	1.68e0	9.91e-1	2.60e0	1.29e0	2.90e0	5.14e0	2.48e0
	25	**0**	0	11.02e0	4.85e0	4.73e0	9.11e0	8.67e0	16.28e0	33.16	11.20
f_7	5	7.34e-2	3.82e-1	2.29e-1	4.32e-1	**0**	0	0	0	7.28e-1	7.06e-1
	10	**1.25e0**	3.13e0	5.22e0	1.07e0	4.37e0	3.04e0	6.12e0	2.84e0	3.63e0	1.57e0
	25	**10.78**	25.49	17.53	5.95	24.75	21.11	45.09	28.98	42.68	14.26
f_8	5	**9.89e-14**	2.02e-12	8.43e-3	8.02e-3	3.84e-2	7.24e-2	2.87e-2	3.93e-2	1.91e-4	6.51e-4
	10	**9.83e-11**	1.99e-9	1.37e-2	1.35e-2	8.33e-2	1.77e-1	4.71e-2	5.37e-2	6.17e-4	2.21e-3
	25	**1.14e-5**	2.55e-4	2.53e-2	2.62e-2	1.24e-1	1.53e-1	6.08e-2	5.87e-2	1.27e-3	5.87e-3

Table 2. 1 : Optimization results.

According to table (2.1), it can be seen that the proposed ETLBO algorithm outperforms the other considered algorithms in 18 cases, while the I-TLBO ($T_n = 2$), I-TLBO ($T_n = 1$) and the original TLBO algorithms give better performances than the proposed ETLBO algorithm in 3 cases, 2cases, and one case, respectively.

Figures (2.3), (2.4) and (2.5) show the convergence speed of each algorithm. It is clear that the proposed ETLBO algorithm has a better convergence speed than the other considered meta-heuristic algorithms, in all considered cases of the dimensions of the optimization problem.

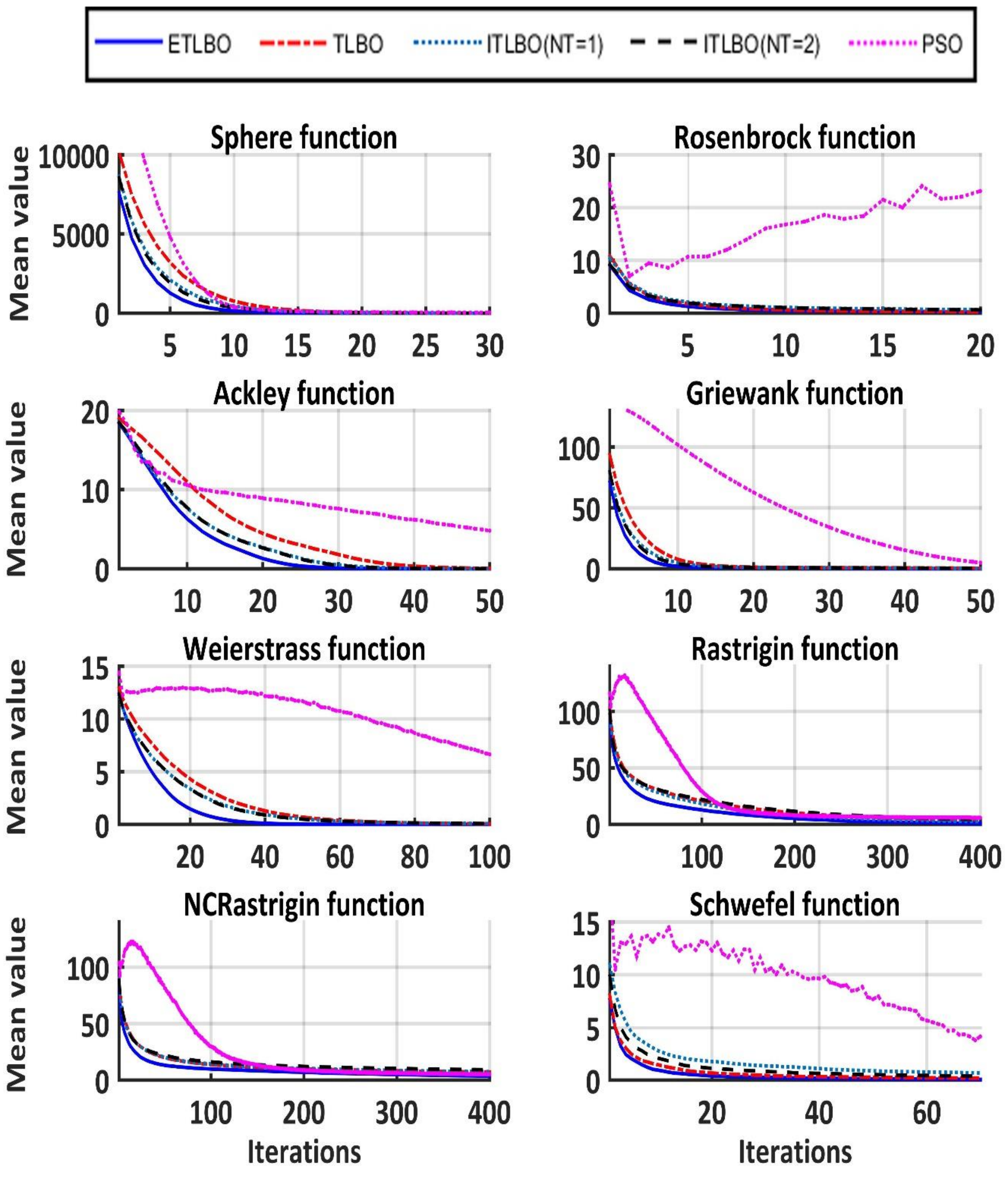


Figure 2. 4 : Convergence speed (D=3 for Rosenbrock function and D=10 for other functions).

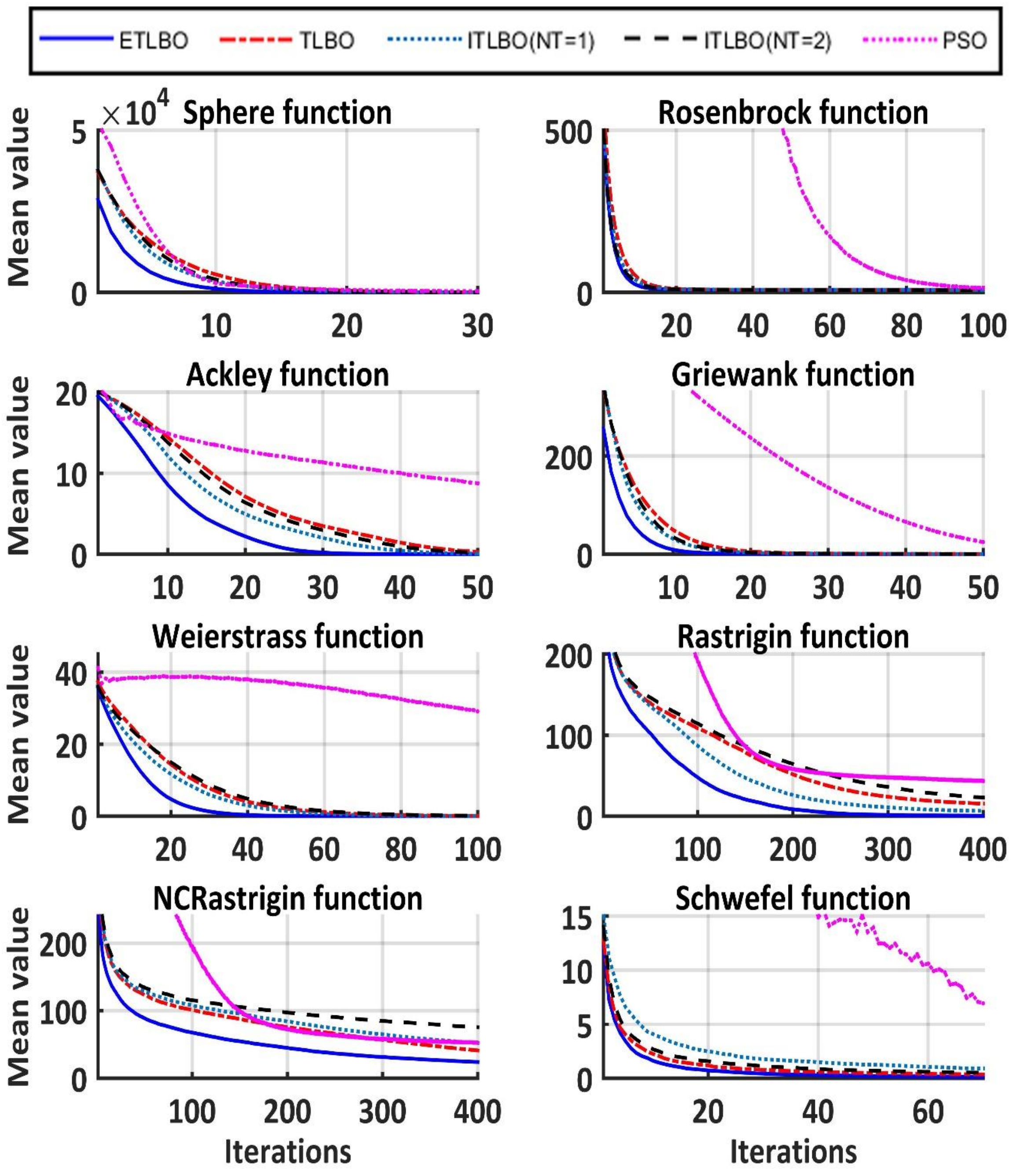


Figure 2. 5 : Convergence speed (D=10 for Rosenbrock function and D=25 for other functions).

6. Conclusion

In this chapter, an enhanced variant of the TLBO algorithm has been proposed. In this algorithm, the random selecting process of learners' pairs was replaced by a more efficient approach based on each student's grade during the optimization process, such that the students only interact with their pairs having a good grade. This modification allows improving the convergence rate and the exploitation quality of the algorithm.

The convergence rate and the efficiency of the proposed algorithm were assessed by considering eight well-known benchmark functions. The obtained results have showed that the proposed ETLBO algorithm outperforms the other considered algorithms; namely the original TLBO, the I-TLBO with one and two teachers, and the w-PSO algorithm.

CHAPTER 3

ARTIFICIAL INTELLIGENT BASED PID CONTROLLERS

1. Introduction

Due to its popularity, simple design procedure, ease of implementation and good control performances, Proportional Integral Derivative (PID) controller is still widely used and the most popular in many industrial applications [183–186]. However, in case of complex highly nonlinear systems, satisfactory control performance cannot be achieved using the conventional linear methods to tune the PID' parameters. Moreover, all linear tuning methods, such as Ziegler-Nicholas technique, gain-phase margin, root locus, minimum variance and gain scheduling, are based on the linearization of the controlled system around the operating point.

In order to achieve good control performance of the conventional PID controller, several methods, which online tune the PID gains to compensate the effect of the modelling uncertainties and the system parameters variation, have been proposed [183,187–189]. These techniques include linear self-tuning methods, non-linear structures of PID controllers, and other methods.

Different types of PID controllers have been proposed to improve the control performance [25,190,191]. Among these types we can find the fractional PID controller [189,192,193], the nonlinear neural network based PID controller [124,194–197], the fuzzy PID controller [198–202], the nonlinear PID controller that uses special nonlinear functions [203], etc.

In the fractional PID controller ($PI^{\lambda}D^{\mu}$), the gains, the integral and the derivative actions are obtained using non-integer integration and differentiation orders [188,189]. The fractional PID controller contains more parameters than the conventional PID. Therefore, by tuning these parameters, good control performance could be achieved. However, computing the fractional integral and derivative is a time consuming and a complex process, especially in real-time applications. Indeed, several methods, to compute the fractional integral and the fractional derivative, such as the Grünwald-Letnikov [204], the Riemann–Liouville and Caputo [205,206] and others [207], have been proposed.

Several other versions of the PID controller, were proposed and used in many real-time applications [208–211]. In fact, these PID versions can be classified according to the technique used to tune the PID gains. Some of them are given as follows:

- Neural Network based PID controller (NNPID) [124,194–196,210,212]: two different architectures of the NNPID can be distinguished. In the first architecture, the connections and the activation functions of the neural network are chosen to mimic the PID. The output of the neural network is exactly the same as the control law of the PID. Therefore, the PID gains are the weights of the neural network and they are optimized using, in most cases, the back propagation algorithm. In the second architecture, the neural network has three outputs; each one calculates one of the PID gains.

- Fuzzy PID controller (FPID) [200,211,213,214]: usually, the control block diagram of the FPID contains three essential blocks; the controlled system, the conventional PID and the fuzzy logic system. The gains of the conventional PID are computed using the fuzzy logic system. Due to its simplicity, the Sugeno type FPID controller is more popular than the Mamdani type.
- Meta-heuristic based PID controller [208,215–223]: using a meta-heuristic algorithm, the parameters of the PID are directly optimized. However, in case of an online optimization process and due to the working principle of meta-heuristic-algorithms, an emulator of the controlled system is required. Several meta-heuristic based PID controllers, such as the PSO based PID controller [198,215,217], the Gray Wolf Optimizer (GWO) based PID controller [218,219], the TLBO based PID controller [220,221], the Artificial Bee Colony (ABC) based PID controller [222,223], have been proposed.

Hybrid versions of PID controllers, that combine neural networks, fuzzy logic systems, and meta-heuristic algorithms, have also been proposed [194,215,224–227].

In this chapter, the adaptive neural network PID (ANNPID) controller, the adaptive Fourier series neural network PID (AFSNNPID) controller and the PSO based PID controller are presented and detailed. To assess the effectiveness of the ANNPID, the AFSNNPID and the PSO based PID controllers, the control of the continuous stirred tank reactor (CSTR) and the 3-DOF robot arm manipulator, through simulation and experimental studies, is considered. In addition, a comparative study is carried out.

2. <u>Adaptive neural network PID controller</u>

The conventional PID control law is given by:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \tag{3.1}$$

where $e(t)$ represents the tracking error and (K_p, K_i, K_d) are the controller gains.

The discrete version of this control law, using the trapezoid method and the sampling period T_s, is given as follows:

$$\begin{aligned} u(k) = u(k-1) + \left(K_p + \frac{K_d}{T_s} + \frac{K_i T_s}{2}\right) e(k) + \\ \left(\frac{K_i T_s}{2} - 2\frac{K_d}{T_s} - K_p\right) e(k-1) + \frac{K_d}{T_s} e(k-2) \end{aligned} \tag{3.2}$$

To compensate the effect of the modeling uncertainties and the system parameters variation and get more satisfactory performance a neural network is used, as it is shown in figure 3.1, to tune the PID gains [124].

The used neural network (figure 3.2) is a feed forward network with three outputs (K_p, K_i, K_d), four inputs $(e(k), e(k-1), u(k-1), u(k-2))$ and one hidden layer.

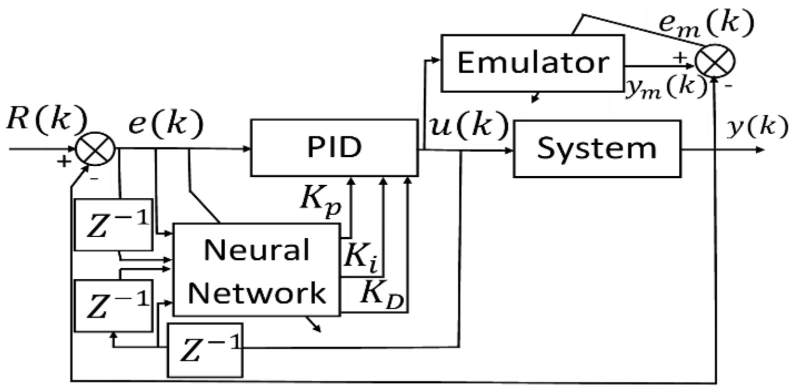


Figure 3. 1 : Adaptive neural network PID controller in closed loop.

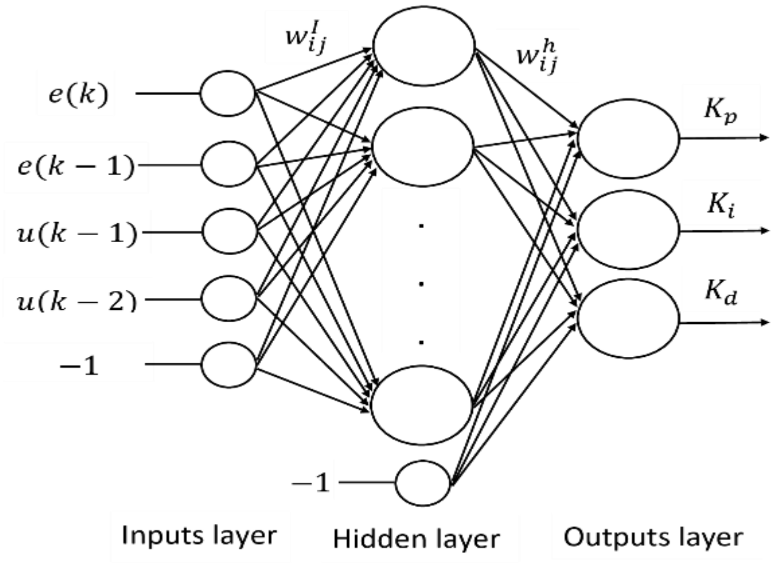


Figure 3. 2 : Structure of the used neural network.

The output of each node of the hidden layer is given by:

$$\begin{cases} \mathrm{h_j} = f(S_j) \\ S_j = \sum_{i=1}^{n+1} w_{ij}^I x_i, \qquad j = 1, \dots, n_\mathrm{h} \end{cases} \tag{3.3}$$

where: ($n = 4$) is the number of inputs, n_h is the number of nodes in the hidden layer, $f(S_j)$ is the sigmoid activation function, w_{ij}^I is the connection weight between the ith neuron of the input layer and the jth neuron of the hidden layer and w_{n+1j}^I is the bias of the jth neuron of the hidden layer and x_i are the inputs of the neural network.

The input vector, at the sampling time k, is given as follows:

$$X(k) = [x_1(k), x_2(k), x_3(k), x_4(k), x_5(k)] = [e(k), e(k-1), u(k), u(k-1), 1]$$

The outputs of the neural network are given by:

$$o_j = \sum_{i=1}^{n_{\mathrm{h}+1}} w_{ij}^h h_i \ , \ j = 1, \dots, n_o \tag{3.4}$$

where: ($n_o = 3$) is the number of outputs, w_{ij}^h is the connection weight between the ith neuron of the hidden layer and the j^{th} neuron of the output layer and $w^{h}_{n_{\text{h}}+1j}$ is the bias of the j^{th} output neuron.

The neural network outputs represent the gains of the PID controller ($k_p = o_1, k_i = o_2, K_d = o_3$). The weights values of the neural network are adjusted so that the objective function given by equation (3.5) is minimized.

$$E = \sum_{k=0}^{N} E(k) \tag{3.5}$$

Such as:

$$E(k) = \frac{1}{2} e(k)^2 = \frac{1}{2}(R(k) - y(k))^2 \tag{3.6}$$

where: $R(k)$ is the reference trajectory and $y(k)$ is the actual system output.

The connections weights of the neural network are adapted, using the back propagation method, according to the following rule:

$$w_{ij}(k+1) = w_{ij}(k) - \eta \, \frac{\partial E(k)}{\partial w_{ij}(k)} \tag{3.7}$$

where: $\eta \in [0\ 1]$ is the learning rate.

For the output layer, the term $\frac{\partial E(k)}{\partial w^{\text{h}}_{ij}(k)}$ can be computed as follows:

$$\frac{\partial E(k)}{\partial w^{\text{h}}_{ij}(k)} = \frac{\partial E(k)}{\partial e(k)} \frac{\partial e(k)}{\partial y(k)} \frac{\partial y(k)}{\partial u(k)} \frac{\partial u(k)}{\partial o_j} \frac{\partial o_j}{\partial w^{\text{h}}_{ij}} \tag{3.8}$$

$$\frac{\partial E(k)}{\partial w^{\text{h}}_{ij}(k)} = -\delta_j(k) \, \frac{\partial o_j}{\partial w^{\text{h}}_{ij}} \tag{3.9}$$

Such as:

$$\delta_j(k) = e(k) \frac{\partial y(k)}{\partial u(k)} \frac{\partial u(k)}{\partial o_j} \ , j = 1, \dots, n_o \tag{3.10}$$

$$\begin{cases} \dfrac{\partial u(k)}{\partial o_1} = e(k) - e(k-1) \\ \dfrac{\partial u(k)}{\partial o_2} = \dfrac{T_s}{2}(e(k) + e(k-1)) \\ \dfrac{\partial u(k)}{\partial o_3} = \dfrac{1}{T_s}(e(k) - 2e(k-1) \ + e(k-2)) \end{cases} \tag{3.11}$$

$$\frac{\partial o_j}{\partial w^{\text{h}}_{ij}} = h_i \,, \qquad i = 1, \dots, n_{\text{h}} + 1, \qquad j = 1, \dots, n_o, \qquad h_{n_{\text{h}}+1} = 1 \tag{3.12}$$

Finally, the adaptation equation can be written as follows:

$$w_{ij}^{\mathrm{h}}(k+1) = w_{ij}^{\mathrm{h}}(k) + \eta\delta_{ij}(k)h_i(k)\,, \qquad i = 1,\dots,n_{\mathrm{h}}+1, \qquad j = 1,\dots,n_o \tag{3.13}$$

For the hidden layer, the gradient of the objective function is obtained as follows:

$$\frac{\partial E(k)}{\partial w_{ij}^{\mathrm{I}}(k)} = \frac{\partial E(k)}{\partial \mathrm{h}_j(k)}\frac{\partial \mathrm{h}_j(k)}{\partial S_j(k)}\frac{\partial S_j(k)}{\partial w_{ij}^{\mathrm{I}}(k)}\,, \qquad i = 1,\dots,n_i+1, \qquad j = 1\dots n_{\mathrm{h}} \tag{3.14}$$

The different terms of equation (3.14) can be computed as follows:

$$\frac{\partial E(k)}{\partial h_j(k)} = -e(k)\frac{\partial y(k)}{\partial u(k)}\sum_{l=1}^{n_o}\frac{\partial u(k)}{\partial \mathrm{o}_l(k)}w_{jl}^{h}(k) \tag{3.15}$$

$$\frac{\partial h_j(k)}{\partial S_j(k)} = h_j(k)(1-h_j(k)) \tag{3.16}$$

$$\frac{\partial S_j(k)}{\partial w_{ij}^{\mathrm{I}}(k)} = x_i(k) \tag{3.17}$$

Equation (3.14) becomes:

$$\frac{\partial E(k)}{\partial w_{ij}^{\mathrm{I}}(k)} = -\sum_{l=1}^{n_o}\delta_l(k)w_{jl}^{h}(k)\,h_j(k)(1-h_j(k))x_i(k) \tag{3.18}$$

where: $\delta_l(k)$ ($l = 1,\dots,n_o$) is given by equation (3.10).

Finally, the adaptation rule is given by:

$$w_{ij}^{\mathrm{I}}(k+1) = w_{ij}^{\mathrm{I}}(k) + \eta\sigma_j(k)x_i(k), \qquad i = 1,\dots,n_o+1, \qquad j = 1,\dots,n_{\mathrm{h}} \tag{3.19}$$

Such as:

$$\sigma_j(k) = h_j(k)\left(1-h_j(k)\right)\sum_{l=1}^{n_o}\delta_l(k)w_{jl}^{h}(k)\,, \qquad j = 1,\dots,n_{\mathrm{h}} \tag{3.20}$$

The term $\frac{\partial y(k)}{\partial u(k)}$ is obtained using an emulator for the system. This emulator is a feed forward multilayer neural network with one input, one hidden layer and one output. The neural network is trained, using the input-output data of the system and the back propagation algorithm, to approximate the system output.

The output $\hat{y}(k)$ of the emulator is given by:

$$\hat{y}(k) = \sum_{i=1}^{m+1} w_i^h(k)o_i^h\,(k) \tag{3.21}$$

where: m is the number of nodes in the hidden layer, w_i^h are the connections weights between the hidden nodes and the output node, w_{m+1}^h is the bias of the output node and o_i^h are the outputs of the hidden nodes, they are given by:

$$\begin{cases} o_j^h(k) = f(s_j^h(k)) \\ s_j^h(k) = w_j^I(k)u(k), j = 1,\dots,m \\ o_{m+1}^h = 1 \end{cases} \tag{3.22}$$

where: w_j^I are the connections weights between the input node and the hidden nodes, $u(k)$ is the input of the system and the emulator, and $f(.)$ is the activation function.

The term $\frac{\partial y(k)}{\partial u(k)}$ of equation (3.8) and (3.15) is approximated as follows:

$$\frac{\partial y(k)}{\partial u(k)} \cong \frac{\partial \hat{y}(k)}{\partial u(k)} = \sum_{i=1}^{m} w_i^h(k) w_i^I(k) o_i^h(k)(1 - o_i^h(k)) \tag{3.23}$$

3. Adaptive Fourier series neural network PID controller

3.1. Fourier series neural network

The FSNN is a Multi Inputs Single Output (MISO) network with one hidden layer, its architecture (figure 3.3) is based on the topological structure of the multidimensional Fourier series. The input layer of the FSNN contains m input nodes receiving the network inputs x_i $(i = 1,2,\dots,m)$. Each input x_i is scaled to the range $[0\ T_i]$ and connected through a fixed frequency weight $\omega_i = \frac{2\pi}{T_i}$ to N_i harmonic neurons in the hidden layer. In addition to the harmonic neurons, there are $l = 2^m$ product nodes in the hidden layer. Each of these nodes implements the product of m outputs of the harmonic neurons. The single linear neuron of the output layer implements the weighted sum of the product nodes outputs. The connections weights $W^j_{n_1,\dots,n_m}$ $(j = 1,\dots,l.\ n_i = 1,\dots,N_i)$ between the hidden layer and the output layer are adapted using an appropriate learning rule.

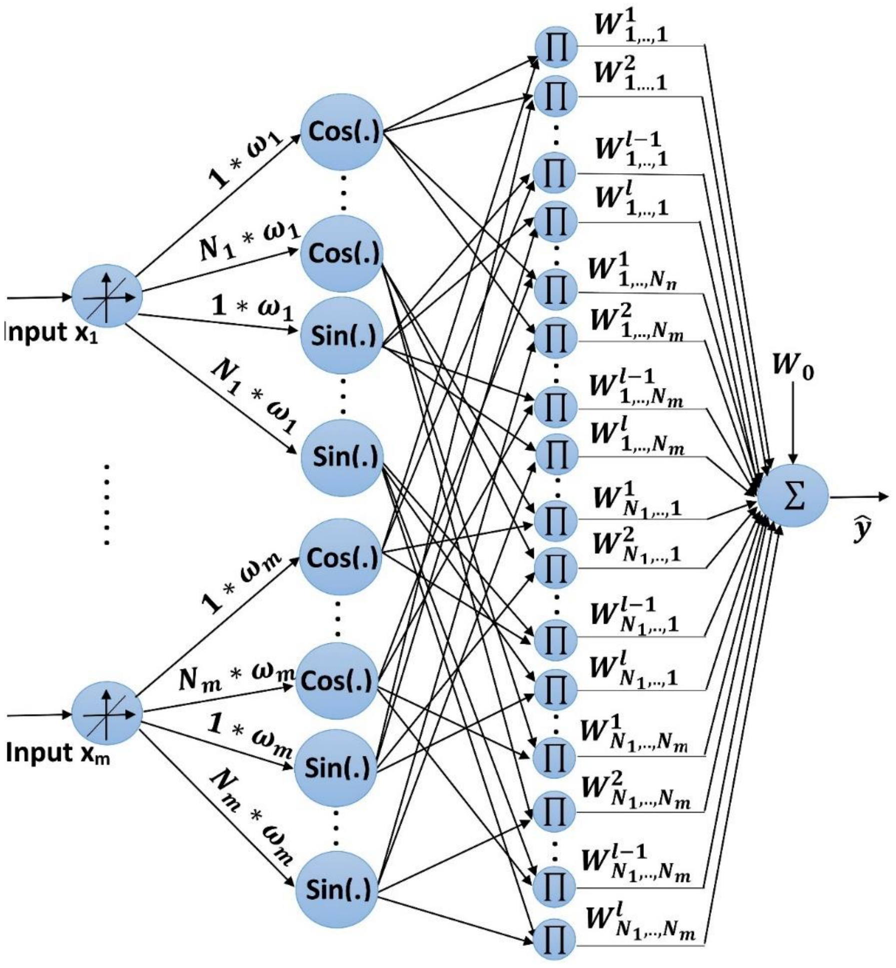


Figure 3. 3 : Fourier series neural network architecture.

The network output is given by:

$$\begin{aligned}\hat{y} = W_0 + \sum_{n_1=1}^{N_1} \dots \sum_{n_m=1}^{N_m} \big[& W^1_{n_1,\dots,n_m} \cos(n_1\omega_1 x_1) \dots \cos(n_m\omega_m x_m) \\ & + W^2_{n_1,\dots,n_m} \cos(n_1\omega_1 x_1) \dots \sin(n_m\omega_m x_m) + \cdots \\ & + W^{l-1}_{n_1,\dots,n_m} \sin(n_1\omega_1 x_1) \dots \cos(n_m\omega_m x_m) \\ & + W^{l}_{n_1,\dots,n_m} \sin(n_1\omega_1 x_1) \dots \sin(n_m\omega_m x_m) \big]\end{aligned} \quad (3.24)$$

where: m is the number of the network inputs, ($X = [x_1, \dots, x_m]$) is the vector of the network inputs, ($\omega_i = \frac{2\pi}{T_i}, i = 1,2, \dots, m$) are the frequency weights, T_i is the range of the input x_i ($x_i \in [0\ T_i]$), ($l = 2^m$) is the number of product nodes, $W^j_{n_1,\dots,n_m}$ is the connections weights (state weights) between the hidden and the output layers, W_0 is the network bias and N_i is the series length.
The number of the weights ($W^j_{n_1,\dots,n_m}$) is given as follows:

$$N_w = l \cdot \prod_{i=1}^{m} N_i \quad (3.25)$$

Equation (3.24) could be rewritten as follows:

$$\hat{y} = W_0 + \sum_{n_1=1}^{N_1} \dots \sum_{n_m=1}^{N_m} \sum_{j=1}^{l} \left[W^j_{n_1,\dots,n_m} H_j \right] \quad (3.26)$$

Such as:

$H_1 = \cos(n_1\omega_1 x_1) \cos(n_2\omega_2 x_2) \dots \cos(n_{m-1}\omega_{m-1} x_{m-1}) \cos(n_m\omega_m x_m)$,

$H_2 = \cos(n_1\omega_1 x_1) \cos(n_2\omega_2 x_2) \dots \cos(n_{m-1}\omega_{m-1} x_{m-1})\ \mathrm{sins}(n_m\omega_m x_m)$,

$$\vdots$$

$H_{l-1} = \sin(n_1\omega_1 x_1) \sin(n_2\omega_2 x_2) \dots \sin(n_{m-1}\omega_{m-1} x_{m-1}) \cos(n_m\omega_m x_m)$,

$H_l = \sin(n_1\omega_1 x_1) \sin(n_2\omega_2 x_2) \dots \sin(n_{m-1}\omega_{m-1} x_{m-1}) \sin(n_m\omega_m x_m)$-

A Multiple Inputs Multiple Outputs (MIMO) FSNN can be created using a set of Multiple Inputs Single Output (MISO) FSNN as shown in figure 3.4.

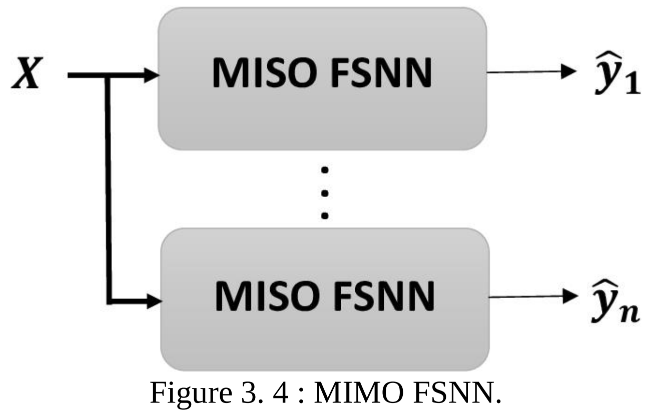


Figure 3. 4 : MIMO FSNN.

The connections weights between the hidden and the output layers are adapted using the Delta rule (gradient descent method) so that a given cost function E is minimized. Let assume that a MIMO system will be modeled using a FSNN strategy. The cost function E is defined as follows:

$$E = \sum_{h=1}^{n} E_h \tag{3.27}$$

Where, n is the number of the system outputs.
At each sampling time k:

$$E_h(k) = \frac{1}{2} e_h(k)^2 = \frac{1}{2}\left(y_h(k) - \hat{y}_h(k)\right)^2 \tag{3.28}$$

Such as:
y_h is the system outputs and $\hat{y}_h$ is the model outputs.
The weights are adjusted according to the following rules:

$$W_0(k) = W_0(k-1) + \Delta W_0(k) \tag{3.29}$$

$$W^j_{n_1,..,n_m}(k) = W^j_{n_1,..,n_m}(k-1) + \Delta W^j_{n_1,..,n_m}(k) \tag{3.30}$$

where:

$$\Delta W_0(k) = -\eta \frac{\partial E(k)}{\partial W_0(k)} \tag{3.31}$$

$$\Delta W^j_{n_1,..,n_m}(k) = -\eta \frac{\partial E(k)}{\partial W^j_{n_1,..,n_m}(k)} \tag{3.32}$$

$j = 1,2,\dots,l.\ \ n_i = 1,2,\dots,N_i$ and $\eta \in [0,1]$ is the learning rate.

The term $\frac{\partial E(k)}{\partial W^j_{n_1,..,n_m}(k)}$ can be computed as follows:

For each output $\hat{y}_h$:

$$\frac{\partial E(k)}{\partial W^j_{n_1,..,n_m}(k)} = \frac{\partial E(k)}{\partial e_h(k)} \frac{\partial e_h(k)}{\partial \hat{y}_h(k)} \frac{\partial \hat{y}_h(k)}{\partial W^j_{n_1,..,n_m}(k)} \tag{3.33}$$

Using equations (3.26, 3.27, 3.28), equation (3.33) becomes:

$$\frac{\partial E(k)}{\partial W^j_{n_1,..,n_m}(k)} = -e_h(k) H_j(k) \tag{3.34}$$

Using equations (3.34), the adaption rules are given by:

$$W^j_{n_1,..,n_m}(k) = W^j_{n_1,..,n_m}(k-1) + \eta e_h(k) H_j(k) \tag{3.35}$$

$$W_0(k) = W_0(k-1) + \eta e_h(k) \tag{3.36}$$

3.2. <u>AFSNNPID controller structure</u>

Two FSNN are used to implement the controller according to the control diagram of figure 3.5 [228]. The FSNN on the right is the emulator FSNN, it is a MISO FSNN that allows emulating the dynamic behavior of the system.

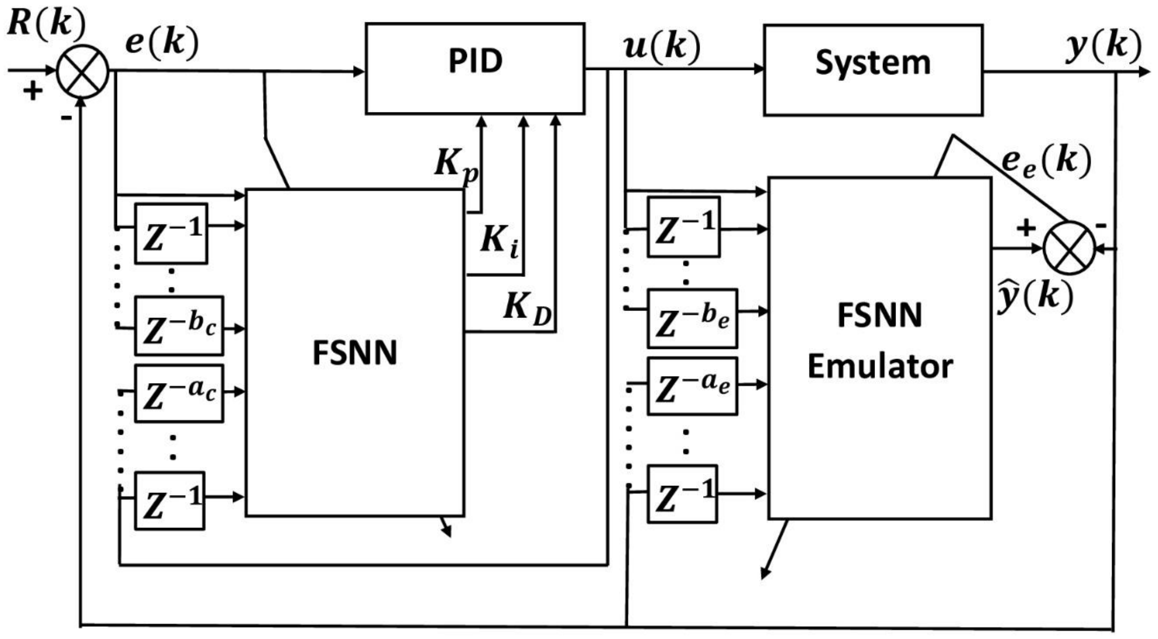


Figure 3. 5 : AFSNNPID controller in closed loop.

The input vector $X_e = [x_1, x_2, \dots, x_m]$ of the emulator FSNN is defined as follows:

$$\mathrm{X_e} = [u(k), u(k-1), \dots, u(k-b_e), y(k-1), y(k-2), \dots, y(k-a_e)]$$

Where, $m_e = 1 + b_e + a_e$ is the number of the FSNN emulator inputs.

The output of the emulator FSNN $\hat{y}$ is given by equation (3.26) and its connections weights are adapted using equations (3.35) and (3.36).

The FSNN on the left is a MIMO FSNN with three outputs (o_1, o_2, o_3). It gives the PID Controller gains such that $o_1 = K_p, o_2 = K_i$ and $o_3 = K_d$. The input vector of this network is given by:

$$\mathrm{X_c} = [e(k), e(k-1), \dots, e(k-b_c), u(k-1), u(k-2), \dots, u(k-a_c)]$$

where $m_c = 1 + b_c + a_c$ is the number of the inputs.

The outputs of the FSNN are given by:

$$o_h = W_0^h + \sum_{n_1=1}^{N_1} \dots \sum_{n_{m_c}=1}^{N_{m_c}} \sum_{j=1}^{l} W_{n_1,..,n_{m_c}}^{j,h} H_j \,, \qquad h = 1:3 \tag{3.37}$$

Such as:

$$H_1 = \cos(n_1\omega_1 x_1)\cos(n_2\omega_2 x_2) \dots \cos\left(n_{m_c-1}\omega_{m_c-1}x_{m_c-1}\right)\cos\left(n_{m_c}\omega_{m_c}x_{m_c}\right)$$

$$H_2 = \cos(n_1\omega_1 x_1)\cos(n_2\omega_2 x_2) \dots \cos\left(n_{m_c-1}\omega_{m_c-1}x_{m_c-1}\right)\sin\left(n_{m_c}\omega_{m_c}x_{m_c}\right)$$

$$\vdots$$

$$H_{l-1} = \sin(n_1\omega_1 x_1)\sin(n_2\omega_2 x_2) \dots \sin\left(n_{m_c-1}\omega_{m_c-1}x_{m_c-1}\right)\cos\left(n_{m_c}\omega_{m_c}x_{m_c}\right)$$

$$H_l = \sin(n_1\omega_1 x_1)\sin(n_2\omega_2 x_2) \dots \sin\left(n_{m_c-1}\omega_{m_c-1}x_{m_c-1}\right)\sin\left(n_{m_c}\omega_{m_c}x_{m_c}\right)$$

where: $W^{j,h}_{n_1,..,n_{m_c}}$ and W^h_0 are the connections weights and the bias of the hth MISO FSNN, respectively.

The FSNN connections weights, giving the PID controller gains, are adapted so that the following objective function is minimized.

$$E(k) = \frac{1}{2}e(k)^2 \tag{3.38}$$

where

$$e(k) = R(k) - y(k) \tag{3.39}$$

The adaption rules are derived, using the Delta rule, as follow:

$$W^h_0(k) = W^h_0(k-1) - \eta\,\frac{\partial E(k)}{\partial W^k_0(k)} \tag{3.40}$$

$$W^{j,h}_{n_1,..,n_{m_c}}(k) = W^{j,h}_{n_1,..,n_{m_c}}(k-1) - \eta\,\frac{\partial E(k)}{\partial W^{j,h}_{n_1,..,n_{m_c}}(k)} \tag{3.41}$$

The term $\frac{\partial E(k)}{\partial W^{j,h}_{n_1,..,n_{m_c}}(k)}$ can be computed as follows:

$$\frac{\partial E(k)}{\partial W^{j,h}_{n_1,..,n_{m_c}}(k)} = \frac{\partial E(k)}{\partial e(k)}\frac{\partial e(k)}{\partial y(k)}\frac{\partial y(k)}{\partial u(k)}\frac{\partial u(k)}{\partial o_h(k)}\frac{\partial o_h(k)}{\partial W^{j,h}_{n_1,..,n_{m_c}}(k)} \tag{3.42}$$

Using equations (3.37 to 3.39), equation (3.42) becomes:

$$\frac{\partial E(k)}{\partial W^{jh}_{n_1,..,n_{m_c}}(k)} = -e(k)\frac{\partial y(k)}{\partial u(k)}\frac{\partial u(k)}{\partial o_h(k)}H_j \tag{3.43}$$

$\frac{\partial u(k)}{\partial o_h(k)}$ is given by equation (3.11).

The term $\frac{\partial y(k)}{\partial u(k)}$, that represents the system Jacobian at time k, is estimated using the FSNN model. To obtain fast convergence and good control performance of the control algorithm, the FSNN model must have a sufficient precision. Of course, a large estimation error could leads to slow convergence or divergence of the control algorithm.

The Jacobian system is obtained as follows:

$$\frac{\partial y(k)}{\partial u(k)} \cong \frac{\partial \hat{y}(k)}{\partial u(k)} = \sum_{n_1=1}^{N_1} \dots \sum_{n_{m_e}=1}^{N_{m_e}} \sum_{j=1}^{l} W^j_{n_1,..,n_{m_e}}(k)\,\frac{\partial H_j(k)}{\partial u(k)} \tag{3.44}$$

Where:

$$\frac{\partial H_1(k)}{\partial u(k)} = -n_1\omega_1 \sin(n_1\omega_1 x_1)\cos(n_2\omega_2 x_2)\dots\cos(n_{m_e-1}\omega_{m_e-1}x_{m_e-1})\cos(n_{m_e}\omega_{m_e}x_{m_e})$$

$$\frac{\partial H_2(k)}{\partial u(k)} = -n_1\omega_1 \sin(n_1\omega_1 x_1)\cos(n_2\omega_2 x_2)\dots\cos(n_{m_e-1}\omega_{m_e-1}x_{m_e-1})\sin(n_{m_e}\omega_{m_e}x_{m_e})$$

$$\vdots$$

$$\frac{\partial H_{l-1}(k)}{\partial u(k)} = n_1\omega_1 \cos(n_1\omega_1 x_1)\sin(n_2\omega_2 x_2)\dots\sin(n_{m_e-1}\omega_{m_e-1}x_{m_e-1})\cos(n_{m_e}\omega_{m_e}x_{m_e})$$

$$\frac{\partial H_l(k)}{\partial u(k)} = n_1\omega_1 \cos(n_1\omega_1 x_1)\sin(n_2\omega_2 x_2)\dots\sin(n_{m_e-1}\omega_{m_e-1}x_{m_e-1})\sin(n_{m_e}\omega_{m_e}x_{m_e})$$

Finally, the adaptation equations can be written as follows:

$$W_0^h(k) = W_0^h(k-1) + \eta \cdot e(k) \frac{\partial y(k)}{\partial u(k)} \frac{\partial u(k)}{\partial o_h(k)} \tag{3.45}$$

$$W_{n_1,..,n_{mc}}^{j,h}(k) = W_{n_1,..,n_{mc}}^{j,h}(k-1) + \eta \cdot e(k) \frac{\partial y(k)}{\partial u(k)} \frac{\partial u(k)}{\partial o_h(k)} H_j \tag{3.46}$$

3.3. Stability analysis

In this section, the stability of the AFSNNPID controller is studied using the small gain theorem. The control system, given in figure 3.5, can be described by the block diagram of figure 3.6, where $u_1(k) = R(k), e_1(k) = e(k), y_1(k) = \Delta u(k), e_2(k) = u(k), y_2(k) = y(k)$ and G_1 and G_2 represent the AFSNNPID controller and the system under control, respectively.

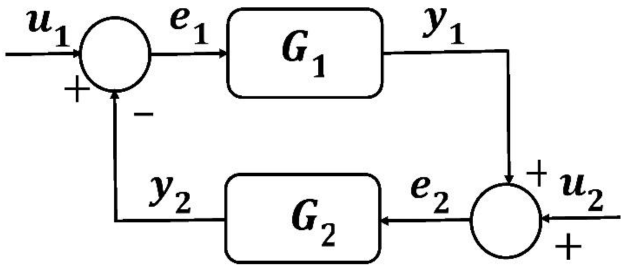


Figure 3. 6 : Feedback control system.

Let suppose that G_1 and G_2 are both stable. According to the small gain theorem, the closed loop system is BIBO stable if $\|G_1\| \cdot \|G_2\| < 1$.

Assuming that the system is modelled using the FSNN model given by equation (3.26), according to equation (3.44), the gain $\|G_2\|$ is given by:

$$\|G_2\| = \sum_{n_1=1}^{N_1} \cdots \sum_{n_{me}=1}^{N_{me}} \sum_{j=1}^{l} \left[W_{n_1,\dots,n_{me}}^{j} \frac{\partial H_j(k)}{\partial u(k)} \right] \tag{3.47}$$

From equation (3.44), we obtain $\frac{\partial H_j(k)}{\partial u(k)} < n_1 \omega_1$, so:

$$\|G_2\| < A \tag{3.48}$$

where:

$$A = \sum_{n_1=1}^{N_1} \cdots \sum_{n_{me}=1}^{N_{me}} \sum_{j=1}^{l} \left[n_1 \omega_1 W_{n_1,\dots,n_{me}}^{j} \right]$$

Using equation (3.2) and (3.37), the controller gain $\|G_1\|$ is given as follows:

$$\begin{aligned}
\|G_1\| &= \big(e(k) - e(k-1)\big) \cdot \sum_{n_1=1}^{N_1} \cdots \sum_{n_{mc}=1}^{N_{mc}} \sum_{j=1}^{l} \left[W_{n_1,\dots,n_{mc}}^{j,h} \frac{\partial H_j(k)}{\partial e(k)} \right], h = 1 \\
\|G_1\| &= \frac{T_s}{2} \big(e(k) + e(k-1)\big) \cdot \sum_{n_1=1}^{N_1} \cdots \sum_{n_{mc}=1}^{N_{mc}} \sum_{j=1}^{l} \left[W_{n_1,\dots,n_{mc}}^{j,h} \frac{\partial H_j(k)}{\partial e(k)} \right], h = 2 \\
\|G_1\| &= \frac{1}{T_s} \big(e(k) - 2e(k-1) + e(k-2)\big) \cdot \sum_{n_1=1}^{N_1} \cdots \sum_{n_{mc}=1}^{N_{mc}} \sum_{j=1}^{l} \left[W_{n_1,\dots,n_{mc}}^{j,h} \frac{\partial H_j(k)}{\partial e(k)} \right], h = 3
\end{aligned} \tag{3.49}$$

Such as:

$$\frac{\partial H_1(k)}{\partial e(k)} = -n_1\omega_1 \sin(n_1\omega_1 x_1)\cos(n_2\omega_2 x_2)\dots\cos\left(n_{m_c-1}\omega_{m_c-1}x_{m_c-1}\right)\cos\left(n_{m_c}\omega_{m_c}x_{m_c}\right)$$

$$\frac{\partial H_2(k)}{\partial e(k)} = -n_1\omega_1 \sin(n_1\omega_1 x_1)\cos(n_2\omega_2 x_2)\dots\cos\left(n_{m_c-1}\omega_{m_c-1}x_{m_c-1}\right)\sin\left(n_{m_c}\omega_{m_c}x_{m_c}\right)$$

$$\vdots$$

$$\frac{\partial H_{l-1}(k)}{\partial e(k)} = n_1\omega_1 \cos(n_1\omega_1 x_1)\sin(n_2\omega_2 x_2)\dots\sin\left(n_{m_c-1}\omega_{m_c-1}x_{m_c-1}\right)\cos\left(n_{m_c}\omega_{m_c}x_{m_c}\right)$$

$$\frac{\partial H_l(k)}{\partial e(k)} = n_1\omega_1 \cos(n_1\omega_1 x_1)\sin(n_2\omega_2 x_2)\dots\sin\left(n_{m_c-1}\omega_{m_c-1}x_{m_c-1}\right)\sin\left(n_{m_c}\omega_{m_c}x_{m_c}\right)$$

It is clear that $\frac{\partial H_j(k)}{\partial e(k)} < n_1\omega_1$, hence we can write:

$$\begin{aligned}
\|G_1\| &= \left(e(k) - e(k-1)\right)\cdot \sum_{n_1=1}^{N_1}\dots\sum_{n_{m_c}=1}^{N_{m_c}}\sum_{j=1}^{l}\left[n_1\omega_1 W^{j,h}_{n_1,\dots,n_{m_c}}\right], h = 1\\
\|G_1\| &= \frac{T_s}{2}\left(e(k) + e(k-1)\right)\cdot \sum_{n_1=1}^{N_1}\dots\sum_{n_{m_c}=1}^{N_{m_c}}\sum_{j=1}^{l}\left[n_1\omega_1 W^{j,h}_{n_1,\dots,n_{m_c}}\right], h = 2\\
\|G_1\| &= \frac{1}{T_s}\left(e(k) - 2e(k-1) + e(k-2)\right)\sum_{n_1=1}^{N_1}\dots\sum_{n_{m_c}=1}^{N_{m_c}}\sum_{j=1}^{l}\left[n_1\omega_1 W^{j,h}_{n_1,\dots,n_{m_c}}\right], h = 3
\end{aligned} \tag{3.50}$$

Using equation (3.48) and (3.50), the term $\|G_1\|\cdot\|G_2\|$ becomes:

$$\begin{aligned}
\|G_1\|\cdot\|G_2\| &< A\left(e(k) - e(k-1)\right)\cdot \sum_{n_1=1}^{N_1}\dots\sum_{n_{m_c}=1}^{N_{m_c}}\sum_{j=1}^{l}\left[n_1\omega_1 W^{j,h}_{n_1,\dots,n_{m_c}}\right], h = 1\\
\|G_1\|\cdot\|G_2\| &< A\frac{T_s}{2}\left(e(k) + e(k-1)\right)\cdot \sum_{n_1=1}^{N_1}\dots\sum_{n_{m_c}=1}^{N_{m_c}}\sum_{j=1}^{l}\left[n_1\omega_1 W^{j,h}_{n_1,\dots,n_{m_c}}\right], h = 2\\
\|G_1\|\cdot\|G_2\| &< \frac{A}{T_s}\left(e(k) - 2e(k-1) + e(k-2)\right)\cdot \sum_{n_1=1}^{N_1}\dots\sum_{n_{m_c}=1}^{N_{m_c}}\sum_{j=1}^{l}\left[n_1\omega_1 W^{j,h}_{n_1,\dots,n_{m_c}}\right], h = 3
\end{aligned} \tag{3.51}$$

In order to ensure the stability of the closed loop system, two conditions must be fulfilled according to the small gain theorem.

The first condition concerns the state weights initialization of the AFSNNPID controller, which is illustrated in equation (3.52).

The second condition is to ensure that the small gain theorem is respected when adapting the controller in real time. Using the Delta rule principle ($W^{j,h}_{n_1,\dots,n_{m_c}}(k) = W^{j,h}_{n_1,\dots,n_{m_c}}(k - 1) - \eta\Delta W^{j,h}_{n_1,\dots,n_{m_c}}$) and equation (3.51), the second condition is given by equation (3.53).

$$
\begin{aligned}
&\sum_{n_1=1}^{N_1} \cdots \sum_{n_{m_c}=1}^{N_{m_c}} \sum_{j=1}^{l} \left[n_1 \omega_1 W_{n_1,\ldots,n_{m_c}}^{j,h}\right] < \frac{1}{A\big(e(k)-e(k-1)\big)}, h = 1 \\
&\sum_{n_1=1}^{N_1} \cdots \sum_{n_{m_c}=1}^{N_{m_c}} \sum_{j=1}^{l} \left[n_1 \omega_1 W_{n_1,\ldots,n_{m_c}}^{j,h}\right] < \frac{2}{AT_s\big(e(k)+e(k-1)\big)}, h = 2 \\
&\sum_{n_1=1}^{N_1} \cdots \sum_{n_{m_c}=1}^{N_{m_c}} \sum_{j=1}^{l} \left[n_1 \omega_1 W_{n_1,\ldots,n_{m_c}}^{j,h}\right] < \frac{T_s}{A(e(k)-2e(k-1)+e(k-2))}, h = 3
\end{aligned}
\tag{3.52}
$$

$$
\begin{aligned}
\eta &< \frac{\dfrac{-1}{A\big(e(k)-e(k-1)\big)} + B^h}{\Delta B^h}, h = 1 \\
\eta &< \frac{\dfrac{-2}{AT_s\big(e(k)+e(k-1)\big)} + B^h}{\Delta B^h}, h = 2 \\
\eta &< \frac{\dfrac{-T_s}{A(e(k)-2e(k-1)+e(k-2))} + B^h}{\Delta B^h}, h = 3
\end{aligned}
\tag{3.53}
$$

where:

$$
B^h = \sum_{n_1=1}^{N_1} \cdots \sum_{n_{m_c}=1}^{N_{m_c}} \sum_{j=1}^{l} \left[n_1 \omega_1 W_{n_1,\ldots,n_{m_c}}^{j,h}\right]
$$

$$
\Delta B^h = \sum_{n_1=1}^{N_1} \cdots \sum_{n_{m_c}=1}^{N_{m_c}} \sum_{j=1}^{l} \left[n_1 \omega_1 \Delta W_{n_1,\ldots,n_{m_c}}^{j,h}\right]
$$

3.4. Control algorithm

Assuming that the FSNN emulator has been trained and the AFSNNPID parameters $(m_c, N_1, N_2, \ldots, N_{m_c}\ and\ \omega_i, i = 1,2,\ldots m_c)$ have been chosen, the proposed AFSNNPID algorithm can be described by the following steps:

Step 0:

- Choose the initial values of the connections weights and the bias of the FSNN giving the PID controller gains such that the conditions (3.52) are satisfied.

Step 1:

- Calculate the error between the reference and the system output using equation (3.39).

Step 2:

- Calculate the approximation of the system Jacobian $\frac{\partial y(k)}{\partial u(k)}$ using equation (3.44).

Step 3:

- For $h = 1:3$
    - Determine the value of η which satisfy the condition given by equation (3.53).
    - Update the bias W_0^h of the control FSNN using equation (3.45).

```
        o For $i = 1: m_c$
            ➢ For $n_i = 1: N_i$
                ✓ For $j = 1: l$
                    ▪ Update the connections weights $W^{j,h}_{n_1,..,n_{m_c}}$ using equation (3.46).
                ✓ End $j$
            ➢ End $n_i$
        o End $i$
    • End $h$
```

Step 4: FSNN emulator adaptation.

```
    • Calculate the error between the system and the FSNN emulator outputs.
    • Update the FSNN model bias $W_0$ using equation (3.36).
    • For $n_i = 1: N_i$
        o For $i = 1: m_e$
            ➢ For $j = 1: l$
                ✓ Update the connections weights $W^{l}_{n_1,..,n_m}$ of the FSNN emulator using equation (3.35).
            ➢ End $j$
        o End $i$
    • End $n_i$
```

Step 5:

- Calculate the value of the control law using equations (3.2, 3.37).
- Apply the obtained control value on the system.
- Wait for the next sampling time, and then go back to step 1.

4. Particle Swarm Optimization based PID controller

In this subsection, the Particle Swarm Optimization based PID (PSO-PID) control algorithm is given. Using the PSO algorithm given in chapter (2), section (4.1) and the control law of the discrete PID controller given by equation (3.2), the gains (K_p, K_i, K_d) of the PID are optimized in order to obtain the best control performance. In fact, these parameters can be optimized offline or during the control process (online) or both.

In general, the offline optimization is sufficient and gives good control performance. However, in the case of severe degradations of the parameters of the controlled system or important external disturbances, the control performance is greatly decreased and in some cases the control loop becomes unstable. This problem is solved using the online optimization, where the PID parameters are optimized during the control process.

4.1. Offline optimization of the PSO based PID controller

Assuming that, the length of the chosen reference trajectory is given by $(samples)$, in most cases, the cost function is chosen as the mean squared error between the desired reference trajectory $(R(k))$ and the output of the system $(y(k))$, it is given by:

$$min_x F(X) = \sum_{k=1}^{samples} \left(R(k) - y(k)\right)^2 \tag{3.54}$$

The optimization algorithm of the PID controller, using the PSO, is given as follows:

Step 0: initialization

- The reference trajectory (R) is chosen by randomly selecting some reference trajectories with random lengths to cover the entire workspace of the controlled system, as follows:

```
o $k = 1$
o For $i = 1: N_r$
    ➢ $r = rand \cdot (y_{max} - y_{min}) + y_{min}$
    ➢ For $j = 1: (rand \cdot L_{max})$
        ✓ $R(k) = r$
        ✓ $k = k + 1$
    ➢ End
o End
```

  Where, y_{max} and y_{min} are the upper limit and the lower limit of the controlled system output, N_r is the number of the reference trajectories and L_{max} is the maximum length of each reference trajectory.
- Randomly generate the initial population (X_i) of the PSO algorithm, using equation (2.3).
- Set the initial velocity ($V_i = 0$), and choose the values of c_1 and c_2.

Step 1: evaluating the entire population

- Evaluate the fitness (F_i) for each particle (X_i), as follows:

```
o For $i = 1: n_p$
    ➢ $F_i = 0$
    ➢ Use $X_i$ as the PID gains
    ➢ For $j = 1: samples$
        ✓ Calculate the control effort using equation (3.2).
        ✓ Apply the control effort to the system input.
        ✓ Calculate the error ($e$) between the system output ($y$) and the desired
          reference trajectory ($R$).
        ✓ Evaluate the cost function as follows: $F_i = F_i + \left(R(j) - y(j)\right)^2$.
    ➢ End
o End
```

Step 2: personal best and global best updating

- Update the values of personal best position (P_i) of each particle and the global best (G) of the entire population, as follows:

```
o For $i = 1: n_p$
    ➢ If $F_i(X_i) < F_i(P_i)$
        ✓ $P_i = X_i$.
        ✓ If $F_i(P_i) < F_i(G)$.
            ▪ $G = P_i$.
        ✓ End if.
    ➢ End if.
o End
```

Step 3: positions updating

- Update the position (X_i) and the velocity (V_i) of each particle personal, as follows:

```
    o For $i = 1{:}\, n_p$
        ➢ Calculate the velocity ($V_i$) using equation (2.4).
        ➢ Calculate the position ($X_i$) using equation (2.5).
    o End
```

Step 4: termination criteria

```
• If $F_i(G) < \varepsilon$ or $k_{max}$ is reached
    o Report the global best solution ($G$).
    o Exit.
• else
    o $k = k + 1$.
    o Go to step 1.
• End
```

4.2. <u>Online optimization of the PSO based PID controller</u>

In the case of the online optimization, a model for the controlled system is required to evaluate the cost function at each sampling time. Several techniques, such as neural network, fuzzy logic, neural fuzzy system … etc, can be used to model the system. The used model must also be adapted online, which complicates the control process and requires a significant computational effort.

In general, at each sampling time (k), the used cost function in the online optimization is given as follows:

$$min_x \, F(X) = \big(R(k) - y(k)\big)^2 \tag{3.55}$$

The control algorithm is given as follows:

Step 0: initialization

- Select the global best position (G), calculated offline, as the initial population (X_i) of the PSO algorithm. If the offline phase has been neglected, randomly generate the initial population (X_i), using equation (2.3).
- Set the initial velocity ($V_i = 0$), and choose the values of c_1 and c_2.

Step 1: reference reading

- Read the current reference point ($R(k)$).

Step 2: optimization

```
• Using the system model, evaluate the fitness ($F_i$) for each particle ($X_i$), as follows:
    o For $i = 1{:}\, n_p$
        ➢ Use $X_i$ as the PID gains.
        ➢ Calculate the control effort using equation (3.2).
        ➢ Apply the control effort to the model input.
        ➢ Calculate the error ($e(k)$) between the model output ($\hat{y}(k)$) and the current
          reference point ($R(k)$).
        ➢ Evaluate the cost function as follows: $F_i = (R(k) - \hat{y}(k))^2$
    o End
• Update the values of each particle personal best position ($P_i$) and the global best ($G$) of
  the entire population, as follows:
        o For $i = 1{:}\,n_p$
            ➢ If $F_i(X_i) < F_i(P_i)$
                ✓ $P_i = X_i$.
                ✓ If $F_i(P_i) < F_i(G)$.
                    ▪ $G = P_i$.
                ✓ End if.
            ➢ End if.
        o End
    • Update the position ($X_i$) and the velocity ($V_i$) of each particle personal, as follows:
        o For $i = 1{:}\,n_p$
            ➢ Calculate the velocity ($V_i$) using equation (2.4).
            ➢ Calculate the position ($X_i$) using equation (2.5).
        o End
    • If $F_i(G) < \varepsilon$ or $k_{max}$ is reached
        o Report the global best solution ($G$).
        o Exit.
    • else
        o Go to step 1.
        o $k = k + 1$.
    • End
```

Step 3: system control

- Use the global best solution (G) as the PID gains.
- Calculate the control effort using equation (3.2).
- Apply the control effort to the system input.
- Calculate the error between the system output and the current reference point.

Step 4: model adaptation

- Apply the control effort that was calculated in step 3, to the model input.
- Calculate the error between the system output and the model output.
- Update the model in order to reduce the error between the system output and the model output.

Step 5: waiting for the next sampling time

- Wait for the next sampling time
- Go back to step 1.

5. Control of the CSTR model

In this Section, the effectiveness of the ANNPID, the AFSNNPID and the PSO based PID controllers are evaluated. The control of a highly nonlinear system, called the Continuous Stirred Tank Reactor (CSTR), is considered.

5.1. System description

The CSTR (figure 3.7) is a highly nonlinear chemical system, where a product A is converted to another product B via an exothermic chemical reaction. The volume v of the reactor is constant, the mixture is considered perfect with a temperature T that is supposed uniform. The reactor operates continuously, its output (the mixture concentration C_a)

changes nonlinearly according to the mixture temperature T, and the temperature changes according to the system input coolant flow q_c.

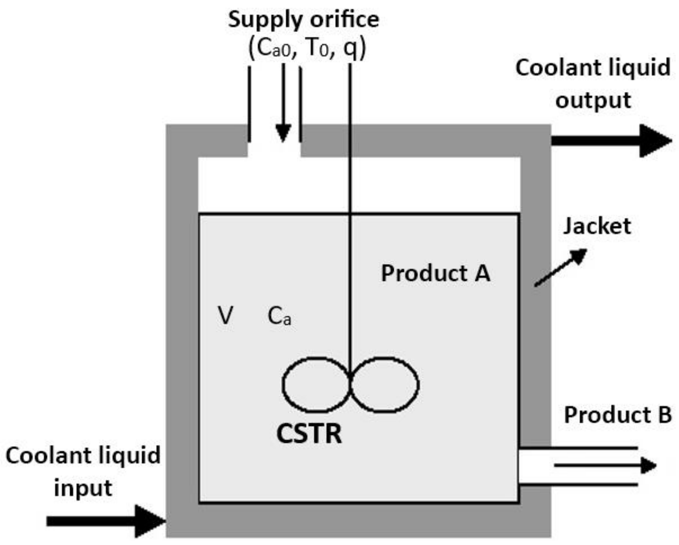


Figure 3. 7 : Continuous stirred tank reactor.

The CSTR is described by the following equation:

$$\dot{C}_a(t) = \frac{q}{v}\big(C_{a0} - C_a(t)\big) - k_0 C_a(t) e^{-\frac{E}{RT(t)}}$$

$$\dot{T}(t) = \frac{q}{v}\big(T_0 - T(t)\big) + k_1 C_a(t) e^{-\frac{E}{RT(t)}} + k_2 q_c(t)\left(1 - e^{-\frac{k_3}{q_c(t)}}\big(T_{c0} - T(t)\big)\right) \tag{3.56}$$

Where,

$$k_1 = -\frac{\Delta H k_0}{\rho C_p}\,, \quad k_2 = \frac{\rho_c C_{pc}}{\rho C_p v}\quad , \qquad k_3 = \frac{h_a}{\rho_c C_{pc}}$$

T_0 is the initial mixture temperature, T_{c0} is the initial jacket temperature, T_c is the jacket temperature and $T(t)$ is the mixture temperature, all temperatures values are expressed in Kelvin. C_{a0} is the initial mixture concentration expressed in mol\l, v is the reactor volume expressed in liter, $q_c(t)$ is the coolant flow expressed in $l\backslash min$ and $C_a(t)$ is the mixture concentration expressed in mol\l.

The constants values are given in table 3.1.

constant	value	constant	value
q	100	ρ	1000
v	100	ρ_c	1000
k_0	7.2e10	C_p	1
E	10000	C_{pc}	1
T_0	350	h_a	700000
T_{c0}	350	C_{a0}	1

Table 3. 1 : CSTR constants values.

5.2. CSTR Modeling

The first step in designing the ANNPID, the AFSNNPID and the PSO based PID controllers, is to obtain a model for the system to be controlled. In the case of the ANNPID and the AFSNNPID controllers, the obtained model is used to estimate the value of the system

Jacobian. In the case of the PSO based PID controller, the obtained model is used to implement the online adaptation of the controller.

Using the CSTR state equations given by (3.56), two different datasets, which cover all possible system working regimes, have been generated. The first one is used to train two different models and the second one is used to test the trained models.
The first model is a static MLP with the following configuration:

- The input layer contains four inputs: $[q_c(k), q_c(k-1), C_a(k-1), C_a(k-2)]$.
- One hidden layer containing 10 neurons with a sigmoid activation function.
- The output layer, which contains a single neuron with a linear activation function, gives the estimated output ($\hat{C}_a(k)$).

This model is implemented with the ANNPID and the PSO-based PID.

The second model is a MISO FSNN with the following configuration:

- The input layer contains two inputs: $[q_c(k), C_a(k-1)]$.
- The hidden layer contains 10 nodes such as: $(N_1 = 5 , N_2 = 5)$.
- The output layer, which contains a single neuron with a linear activation function, gives the estimated output ($\hat{C}_a(k)$).

This model is implemented with the AFSNNPID controller.

Figure 3.8 and 3.9 show the responses of the obtained models and the system using the second dataset. Table 3.2 gives the values of the Root Mean Square of the modeling Error (RMSE) and the correlation coefficient (R^2), for both models.

	NN model	FSNN model
RMSE	6.909e-5	1.549123e-04
R^2	0.99999943	0.99999367

Table 3. 2 : RMSE and R^2 for both models.

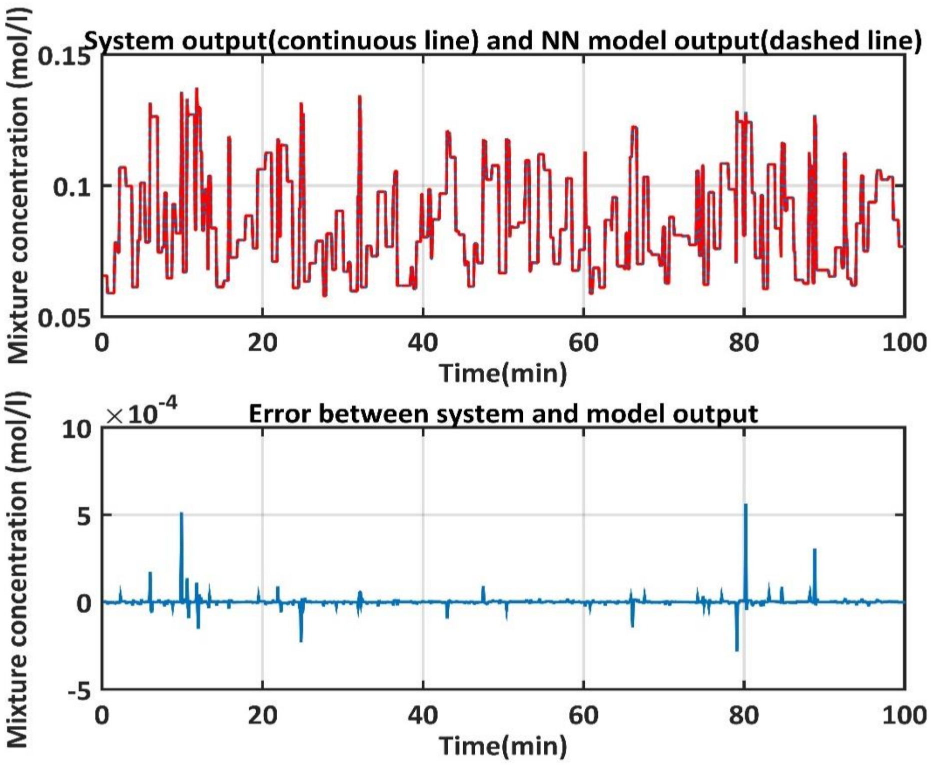


Figure 3. 8 : Test results of the obtained NN model.

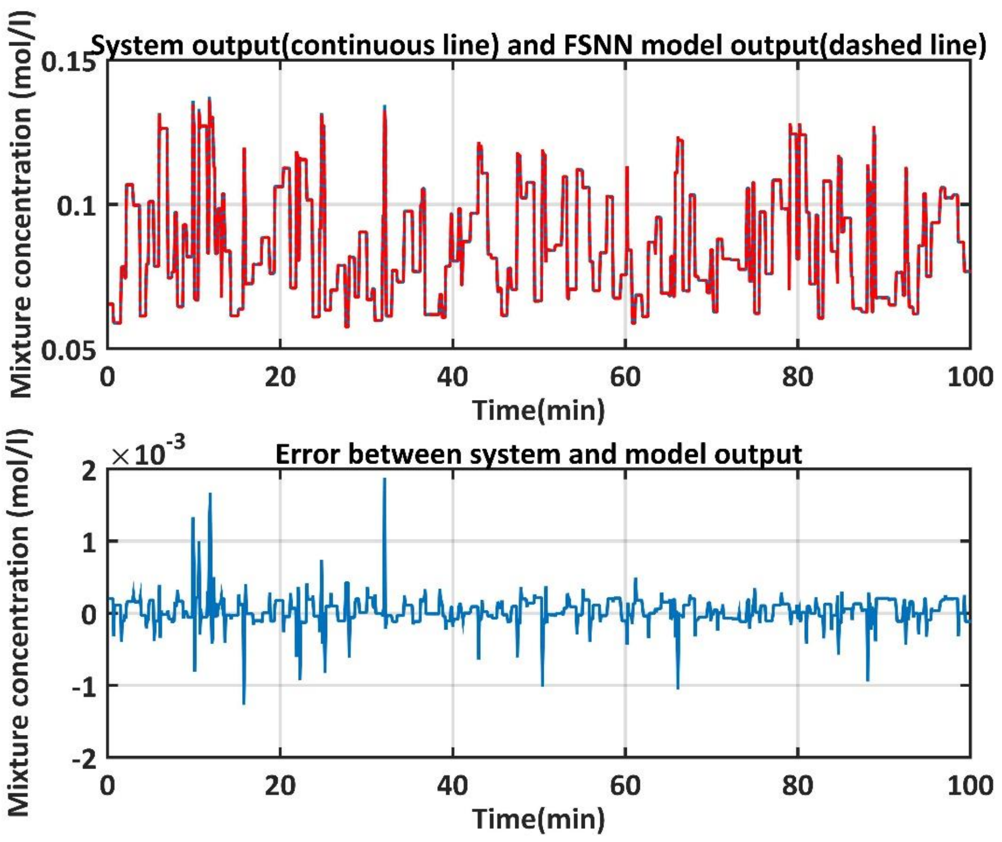


Figure 3. 9 : Test results of the obtained FSNN model.

From figure (3.8) and (3.9), it can be seen that the models output and the system output are superposed, and the modeling error is quite small for both models. According to table (3.2), it can be concluded that the obtained models are quite accurate. Therefore, both models are validated.

5.3. Controllers implementation

In this subsection, the architecture of the ANNPID, the AFSNNPID and the PSO-based PID, are described and given.

5.3.1. ANNPID

In this architecture, one MLP neural network is used to determine the gains K_p, K_i, K_d of the PID controller as shown in figure (3.10). The architecture of the neural network has the following configuration:

- The input layer contains four inputs: $[e(k), e(k-1), q_c(k-1), q_c(k-2)]$.
- One hidden layer containing 10 neurons with a sigmoid activation function.
- The output layer, which contains three neurons with a linear activation function, gives the values of the PID gains (K_p, K_i, K_d).

The weights of this NN are trained and adapted online using equations (3.13) and (3.19). The CSTR neural model, obtained in the previous section, is used to estimate the system Jacobian required to implement the control algorithm, according to equation (3.23).

5.3.2. AFSNNPID

Three FSNN having the same input $e(k)$ (the error between the reference trajectory and the system output), with a range of $T = 0.3$, are used to determine the gains K_p, K_i, K_d of the

PID controller as shown in figure (3.11). The three networks have the same number of hidden nodes ($N = 5$). The weights of these three FSNN are trained and adapted online using equations (3.45) and (3.46). The system Jacobian is estimated using the CSTR FSNN model obtained in the previous section according to equation (3.44).

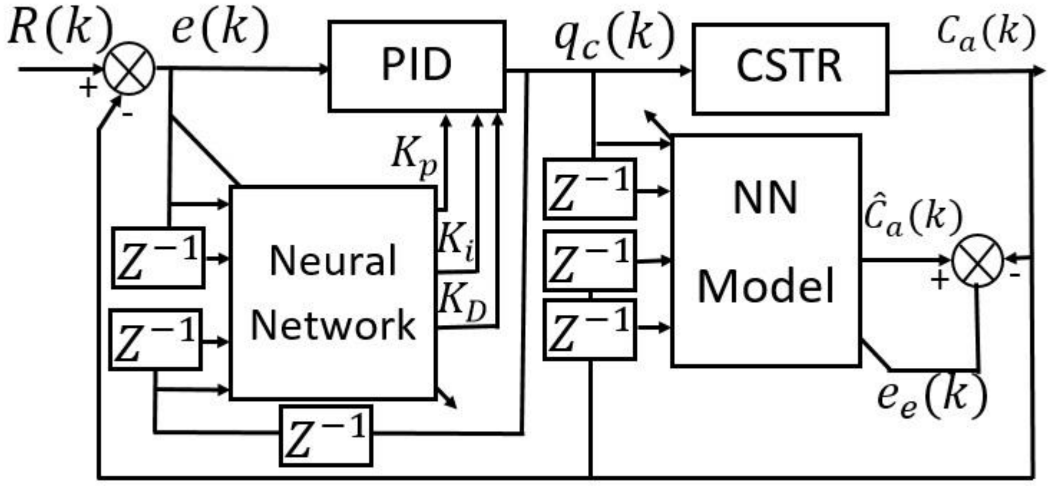


Figure 3. 10 : Control loop of the CSTR model using the ANNPID controller.

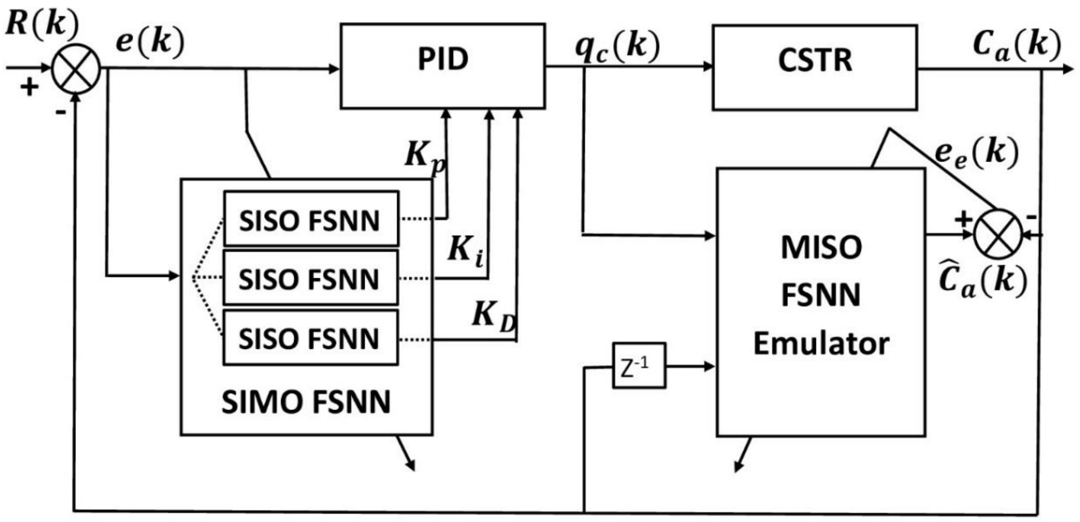


Figure 3. 11 : Control loop of the CSTR model using the AFSNNPID controller.

5.3.3. PSO-based PID

The control law of the used PID controller is given by equation (3.2), where the PSO algorithm is implemented to optimize offline and online the PID gains according to the algorithms given in subsection (4.1) and (4.2), respectively. The cost function is given by equation (3.55). The control block diagram is given by figure (3.12). The chosen parameters of the PSO algorithm that were used to offline and online optimize the PID gains are given by table 3.3.

	c_1	c_2	k_{max}	ε	ω_d
Offline optimization	2	2	100000	1e-5	0.99
Online optimization	2	2	10	1e-7	0.99

Table 3. 3 : Parameters values of the PSO algorithm.

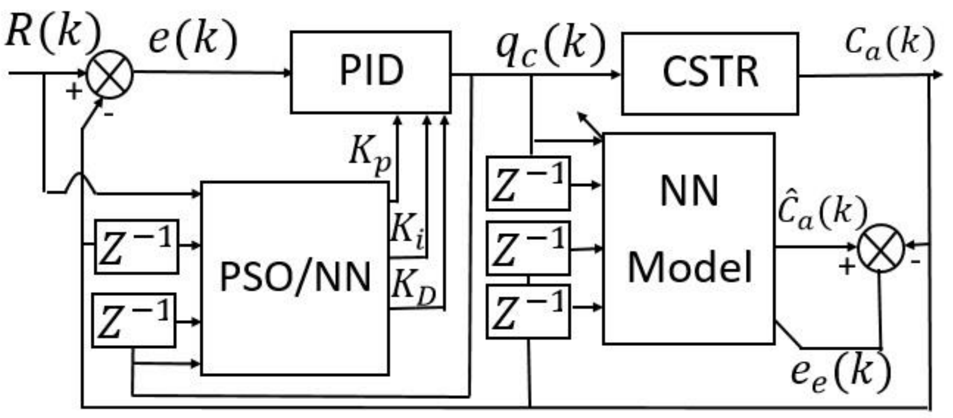


Figure 3. 12 : Control loop of the CSTR model using the PSO-based PID controller.

5.4. Simulation Results

To highlight the control performance of the ANNPID, the AFSNNPID and the PSO-based PID controllers, a comparative study of the abovementioned controllers, considering various operating conditions, is carried out. In the first simulation, we have used a reference trajectory composed of three steps having duration of 4 min and amplitudes of 0.08, 0.1 and 0.12 mol/l, respectively. Figure (3.13) shows the obtained control results. Table 3.4 gives the MSE, MAE and RMSE values computed over the time interval of the simulation.

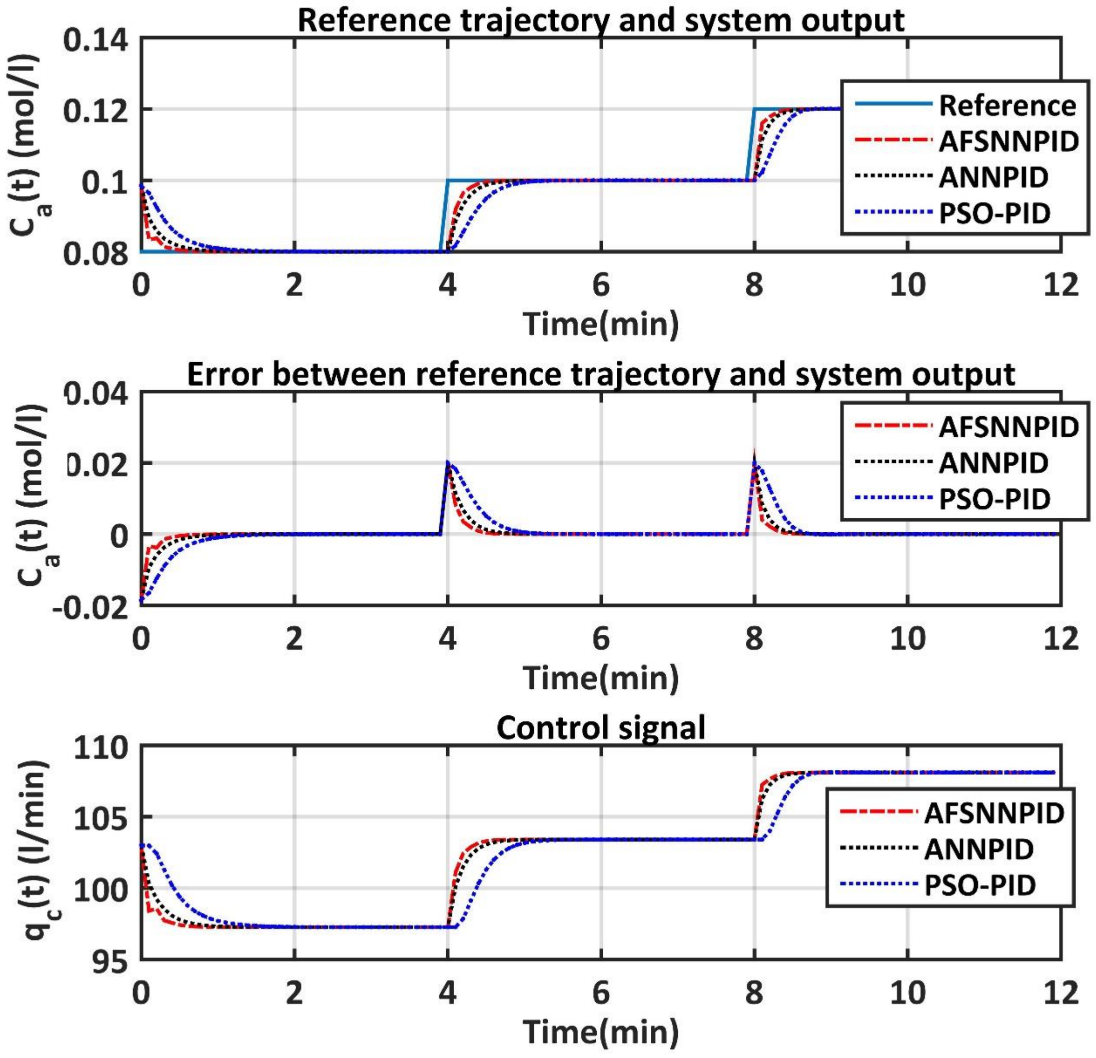


Figure 3. 13 : Control results for the case of multistep reference trajectory.

	AFSNNPID	ANNPID	PSO-PID
MSE $(\cdot 10^{-5})$	1.0605	1.3137	2.5413
MAE $(\cdot 10^{-3})$	0.7637	1.0616	1.9207
RMSE$(\cdot 10^{-3})$	3.2566	3.6245	5.0411

Table 3. 4 : MSE, MAE and RMSE in the case of multistep reference trajectory.

From figure (3.13) and table (3.4), it can be seen that a good tracking accuracy of the reference trajectory is obtained for all implemented controllers. However, the proposed AFSNNPID controller has better control performance, in terms of the tracking accuracy and the settling time, than the other controllers.

The aim of the second simulation is to assess the control performance when using a sinusoidal reference trajectory with a magnitude of 0.02 (mol/l), a bias of 0.1 (mol/l) and a frequency of 1/120 (Hz). Figure (3.14) shows the obtained control results. Table 3.5 gives the MSE, MAE and RMSE values computed over the time interval of the simulation.

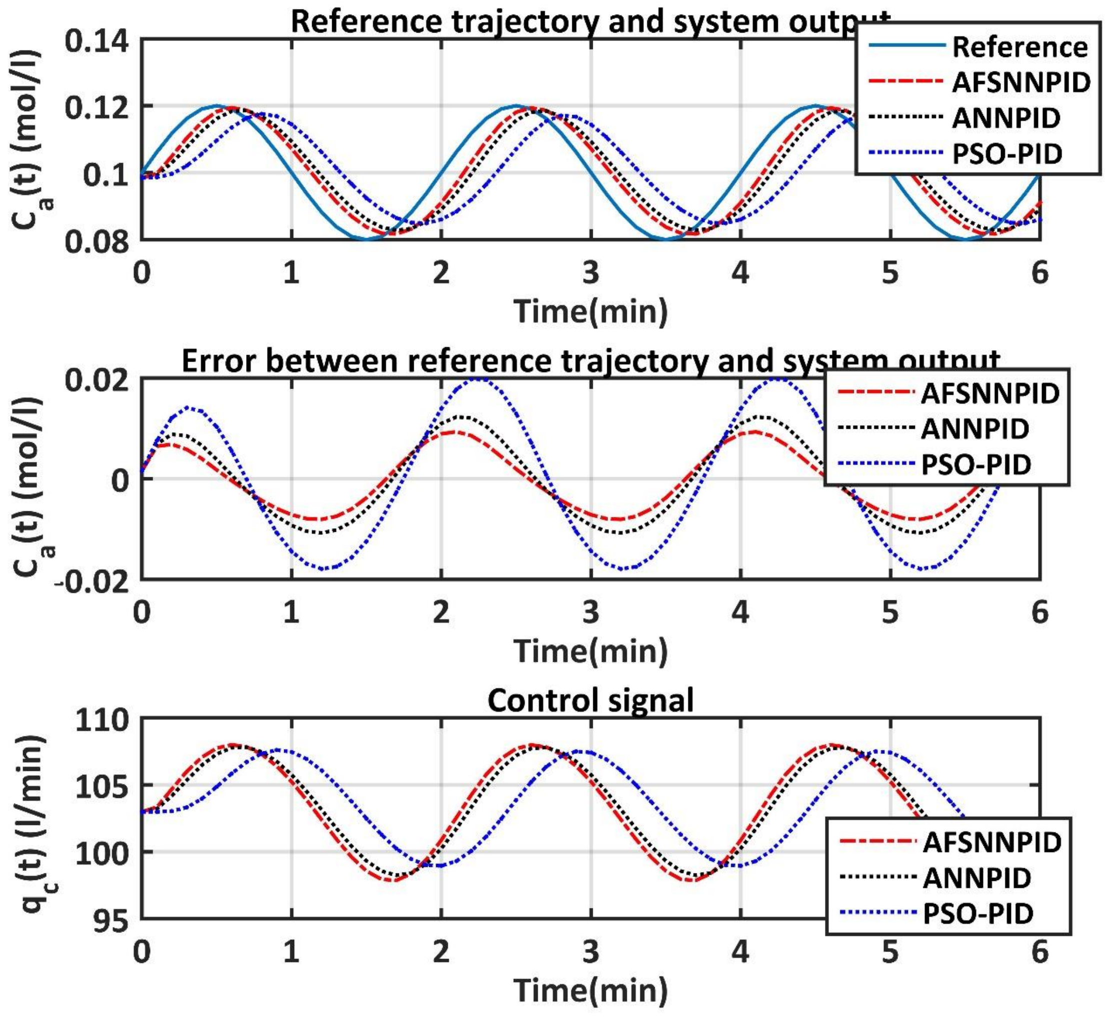


Figure 3. 14 : Control results for the case of sinusoidal reference trajectory.

	AFSNNPID	ANNPID	PSO-PID
MSE $(\cdot 10^{-5})$	3.4897	6.1121	16.3442
MAE $(\cdot 10^{-3})$	5.2729	6.9953	11.4289
RMSE$(\cdot 10^{-3})$	5.9073	7.8180	12.7845

Table 3. 5 : MSE, MAE and RMSE in the case of sinusoidal reference trajectory.

According to figure (3.14) and table (3.5), it can be seen that a good tracking accuracy of the sinusoidal reference trajectory is obtained for both AFSNNPID and ANNPID controllers. However, in the case of the PSO based PID controller, degradation in the control performance is observed, in particular in terms of the settling time. Also, in this simulation, the proposed AFSNNPID controller was found to have better control performance, in terms of tracking accuracy and stabilization time, than the other controllers.

In order to evaluate the robustness of these controllers against external disturbances in the input, in the third simulation, a step reference trajectory with a duration of 10 min and an amplitude of 0.11 (mol/l), was used. During the control process, a leak in the coolant flow q_c (CSTR input) with an amplitude of 5 $(l\backslash min)$ in the time interval [4min 7min], is added. The obtained control results are given in figure (3.15). The MSE, MAE and RMSE values are computed over the time interval of the simulation and given in table (3.6).

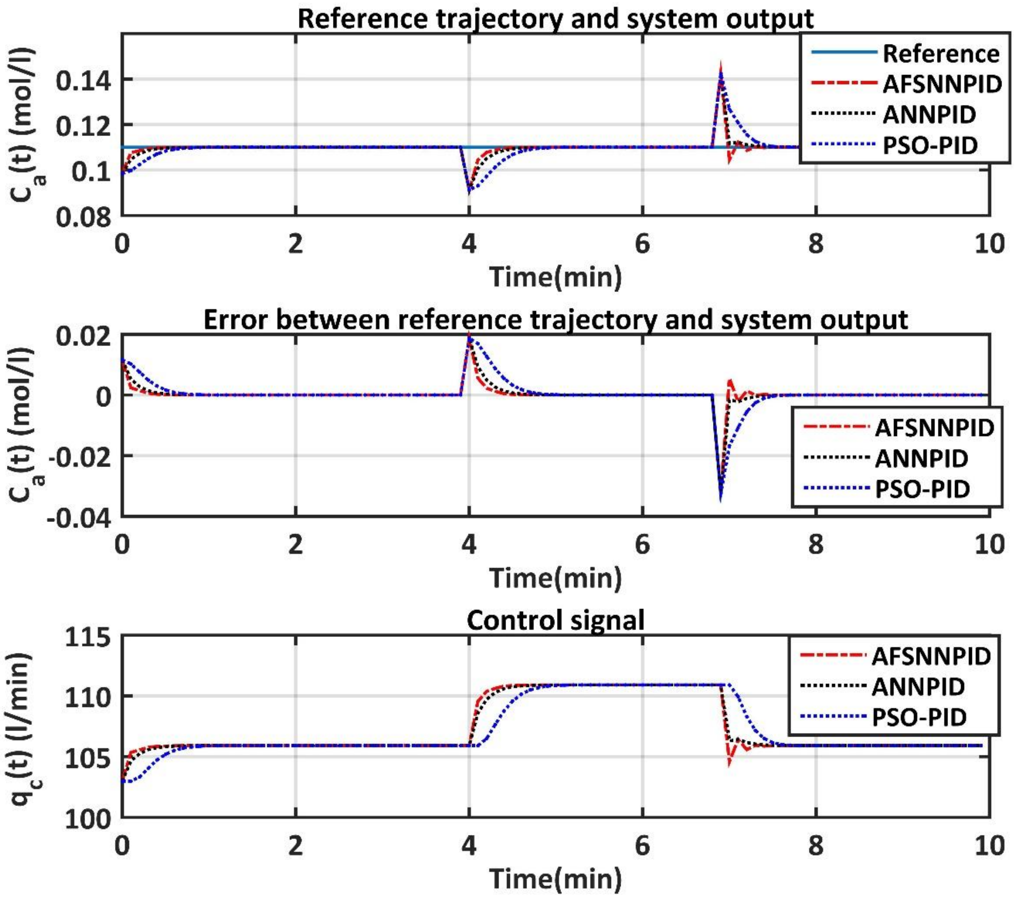


Figure 3. 15 : Control results in the presence of an input disturbance.

	AFSNNPID	ANNPID	PSO-PID
MSE $(\cdot 10^{-5})$	1.6211	1.7068	2.7426
MAE $(\cdot 10^{-3})$	0. 8690	0. 9945	1.7972
RMSE$(\cdot 10^{-3})$	4.0263	4.1314	5.2369

Table 3. 6 : MSE, MAE and RMSE in the presence of an input disturbance.

From figure (3.15) and table (3.6), it can be concluded that the three controllers can compensate very quickly the input leakage and a good tracking accuracy of the reference trajectory is obtained, even during the perturbation. Hence, it can be concluded that these controllers are robust against input disturbances. However, in the case of the AFSNNPID, the response time needed to compensate the input disturbance is smaller than that obtained using the ANNPID and the PSO based PID controllers.

In the last simulation, the robustness of the considered controllers against output disturbances is evaluated. The chosen reference trajectory is exactly the same as that used in the third simulation. Although, in this simulation, during the control process, a leak in the mixture concentration C_a with an amplitude of 0.011 $(mol\backslash l)$ in the time interval [3min 7min], is added. The obtained control results are given by figure (3.16). The MSE, MAE and RMSE values are computed over the time interval of the simulation and given by table (3.7).

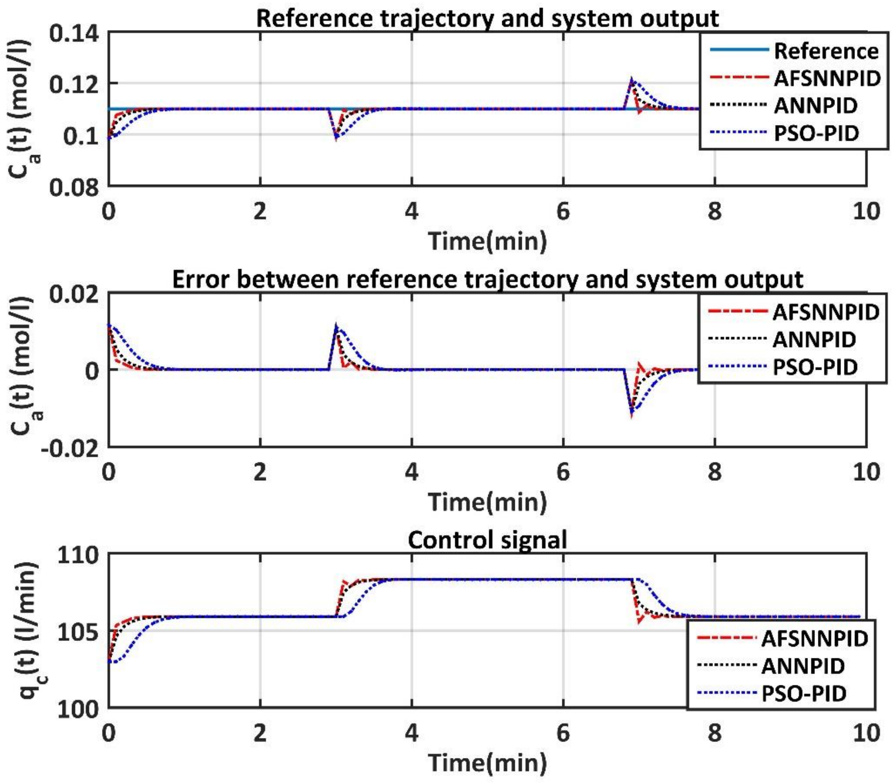


Figure 3. 16 : Control results in the presence of an output disturbance.

	AFSNNPID	ANNPID		PSO-PID
MSE $(\cdot 10^{-6})$	3.9231	4.5205		8.7088
MAE $(\cdot 10^{-3})$	0. 4475	0. 5944		1.0777
RMSE$(\cdot 10^{-3})$	1.9807	2.1261		2.9511

Table 3. 7 : MSE, MAE and RMSE in the presence of an output disturbance.

From figure (3.16) and table (3.7), it can be seen that the output disturbance is compensated for all considered controllers. We notice that the AFSNNPID required a smaller time to compensate the output disturbance than the ANNPID and the PSO based PID controllers.

6. Control of a 3-DOF robot arm manipulator

6.1. Experimental Setup

To demonstrate the effectiveness of the AFSNNPID, the ANNPID and the PSO based PID controllers, the control of the three degrees of freedom (3-DOF) robot arm manipulator, shown in the experimental setup of figure (3.17), is considered. Two 24V 12RPM DC motors are used to generate the rotational movements of joint 1 and joint 2 of this manipulator, and the 12V 170RPM DC motor is used to vertically move its electromagnet end effector. The structure of the considered 3-DOF robot arm manipulator is shown in figure (3.18), the parameters values of the considered manipulator are gathered in table (3.8). In this experimental setup, three H-bridges DC motor drivers, the electromagnet MOSFET-based switch and the DC power supply are carried out and used. The control algorithm is implemented using the DSP board TMS320F28335.

parameter	value	parameter	value
m_1	2.3 (kg)	l_1	0.138 (m)
m_2	0.6 (kg)	l_2	0.1965 (m)
m_3	2.36 (kg)	l_3	0.34 (m)
d_1	0.175 (m)	d_2	0.165 (m)

Table 3. 8 : Parameters values of the considered manipulator.

In order to calculate the joint coordinates for a given set of end effector coordinates, the inverse kinematics model, given by the following equations, is used:

$$\theta_2 = \text{acos}\left(\frac{(x^2+y^2)-(L_1^2+L_2^2)}{2*L_1*L_2}\right)$$
$$\theta_1 = atan2(y,x) - atan2(L_2*\sin(\theta_2), L_1+L_2*\cos(\theta_2)) \tag{3.57}$$

where: x, y are the end effector coordinates (expressed in meter), θ_1, θ_2 are the joints coordinates (expressed in rad), and L_1 and L_2 are the first link length and the second link length (expressed in meter), respectively.

The control block diagram is given in figure 3.19, where: θ_1 and θ_2 are the angles of joint 1 and joint 2 respectively, z is the end effector altitude, u_1, u_2, u_3 are the calculated control voltages, u_{em} is the applied voltage on the electromagnet end effector, K_{p_j}, K_{i_j} and K_{d_j} for $j = 1,2,3$ are the gains of the PID controllers, R_{θ_1}, R_{θ_2} and R_z are the reference trajectories for θ_1, θ_2 and z respectively.

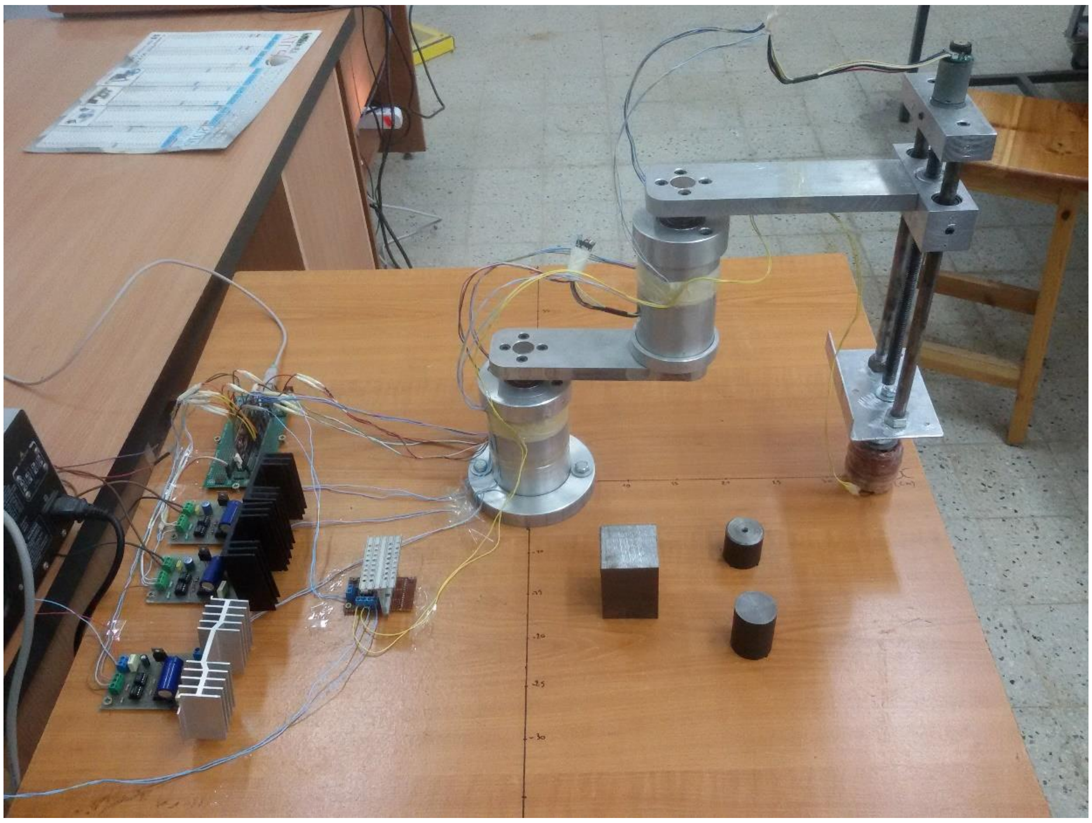


Figure 3. 17 : 3-DOF robot arm manipulator experimental setup.

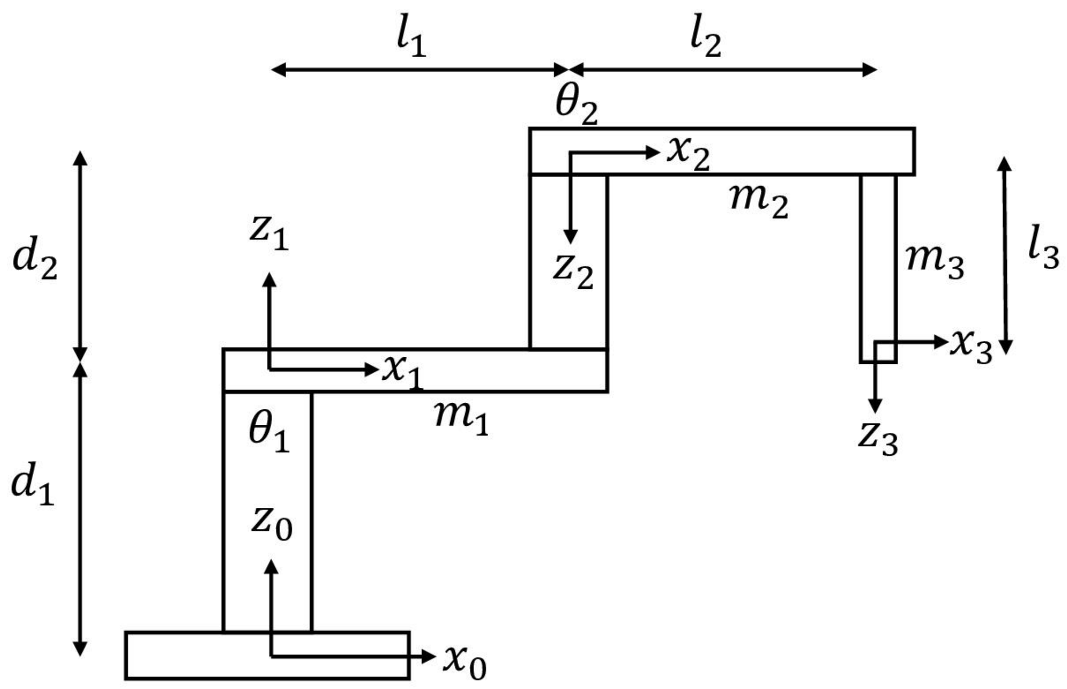


Figure 3. 18 : 3-DOF robot arm manipulator diagram.

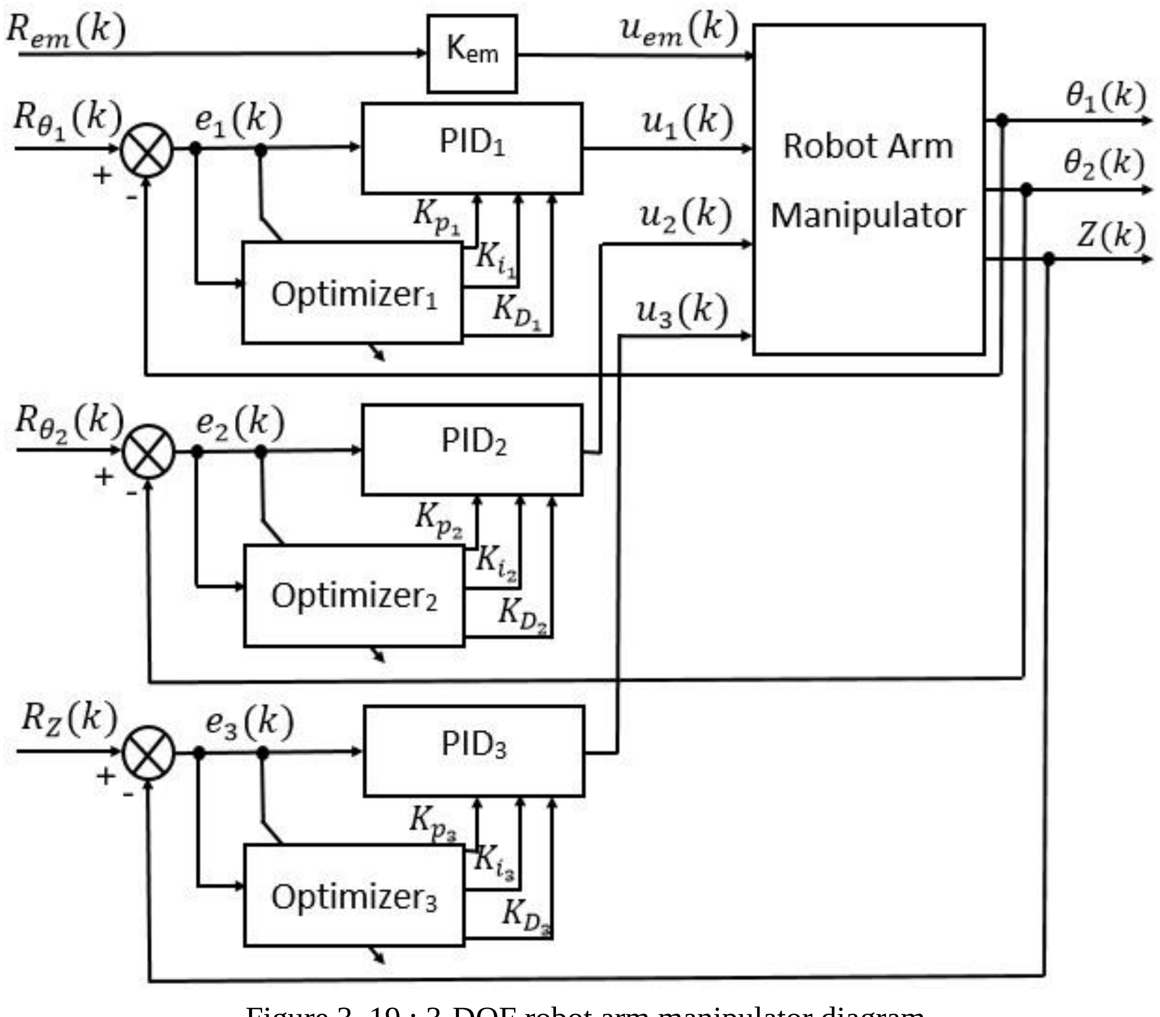


Figure 3. 19 : 3-DOF robot arm manipulator diagram.

The block, named optimizer, can be an FSNN, a NN or a PSO algorithm depending on the implemented controller (AFSNNPID, ANNPID or PSO-PID).

6.2. Robot arm manipulator modeling

As stated earlier, the first step in designing the ANNPID, the AFSNNPID and the PSO based PID controllers, is obtaining a model for the 3-DOF robot manipulator. In the case of the ANNPID and the AFSNNPID controllers, the obtained model is used to estimate the value of the system Jacobian. In the case of the PSO based PID controller, the obtained model is used to implement the online adaptation of the controller. By applying random voltages to the inputs of the manipulator (u_1, u_2, u_3) and measuring the corresponding outputs (θ_1, θ_2, z), two different datasets, which cover all possible system working regimes, have been generated. The first is used to train the FSNN model and the NN model, the second is used to test the trained models.

The NN model is a set of three MISO MLPs, $(\theta_1 - NN_{model}, \theta_2 - NN_{model}, z - NN_{model})$.

- The $(\theta_1 - NN_{model})$ has the following architecture:
  - The input layer contains four inputs: $[u_1(k), u_1(k-1), \theta_1(k-1), \theta_1(k-2)]$.
  - One hidden layer containing 8 neurons with a sigmoid activation function.
  - The output layer, which contains a single neuron with a linear activation function, gives the estimated output ($\hat{\theta}_1(k)$).
- The $(\theta_2 - NN_{model})$ has the following architecture:
  - The input layer contains four inputs: $[u_2(k), u_2(k-1), \theta_2(k-1), \theta_2(k-2)]$.
  - One hidden layer containing 8 neurons with a sigmoid activation function.
  - The output layer, which contains a single neuron with a linear activation function, gives the estimated output ($\hat{\theta}_2(k)$).
- The $(z - NN_{model})$ has the following architecture:
  - The input layer contains four inputs: $[u_3(k), u_3(k-1), z(k-1), z(k-2)]$.
  - One hidden layer containing 4 neurons with a sigmoid activation function.
  - The output layer, which contains a single neuron with a linear activation function, gives the estimated output ($\hat{z}(k)$).

The FSNN model is a set of three MISO FSNN, $(\theta_1 - FSNN_{model}, \theta_2 - FSNN_{model}, z - FSNN_{model})$.

- The $(\theta_1 - FSNN_{model})$ has the following architecture:
  - The input layer contains two inputs: $[u_1(k), \theta_1(k-1)]$.
  - The hidden layer contains 10 nodes such as: ($N_1 = 5$, $N_2 = 5$).
  - The output layer, which contains a single node with a linear activation function, gives the estimated output ($\hat{\theta}_1(k)$).
- The $(\theta_2 - FSNN_{model})$ has the following architecture:
  - The input layer contains two inputs: $[u_2(k), \theta_2(k-1)]$.
  - The hidden layer contains 10 nodes such as: ($N_1 = 5$, $N_2 = 5$).
  - The output layer, which contains a single node with a linear activation function, gives the estimated output ($\hat{\theta}_2(k)$).
- The $(z - FSNN_{model})$ has the following architecture:
  - The input layer contains two inputs: $[u_3(k), z(k-1)]$.
  - The hidden layer contains 6 nodes such as: ($N_1 = 3$, $N_2 = 3$).
  - The output layer, which contains a single node with a linear activation function, gives the estimated output ($\hat{z}(k)$).

Figure (3.20) and (3.21) show the responses of the obtained models and the system using the second dataset. Table (3.9) gives the values of the RMSE and the correlation coefficient (R^2), for both models.

	RMSE	R^2
$\theta_1 - FSNN_{model}$	0.00706022581570082	0.999999979345815
$\theta_2 - FSNN_{model}$	0.222668777573781	0.999983536274307
$Z - FSNN_{model}$	0.00521404903853009	0.999998065136512
$\theta_1 - NN_{model}$	0.00838143327553178	0.999999970889062
$\theta_2 - NN_{model}$	0.721650679061039	0.999827751814977
$Z - NN_{model}$	0.0432449726288857	0.999866805079681

Table 3. 9 : RMSE and R^2 for both 3-DOF manipulator models.

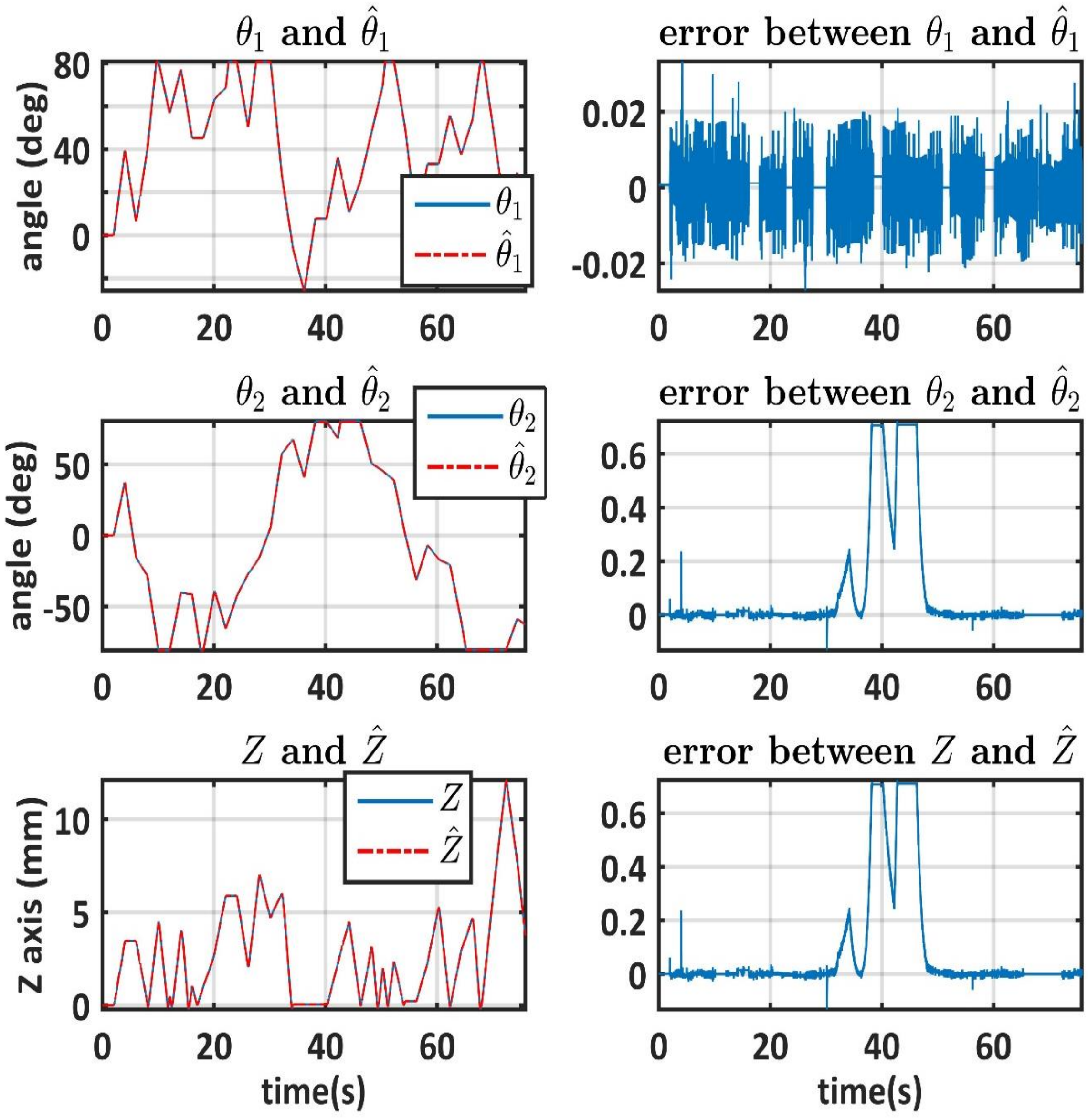


Figure 3. 20 : Test results of the obtained 3-DOF manipulator NN model.

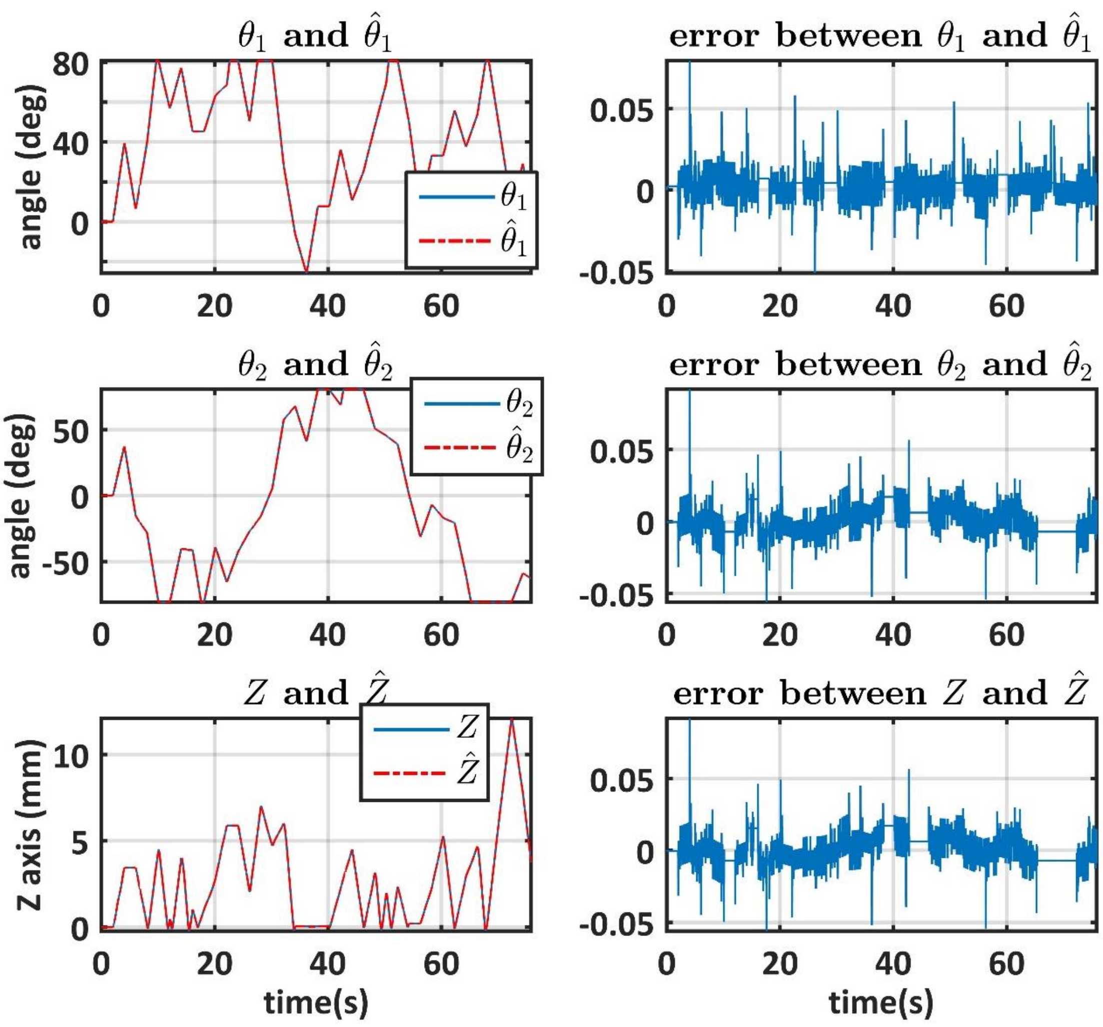


Figure 3. 21 : Test results of the obtained 3-DOF manipulator FSNN model.

From figure (3.20) and (3.21), it can be seen that the models output and the system output are superposed, and the modeling error is quite small for all obtained models. According to table (3.9), it can be concluded that the obtained models are quite accurate.

6.3. Experimental results

The ANNPID, the POS based PID and the AFSNNPID controllers are coded and implemented on the TMS320F28335 DSP board using a sampling period $T_s = 10ms$, the FSNN and NN models previously obtained. The PWM (Pulse Width Modulation) signals required to drive the H-bridges are generated using this board, with the appropriate duty cycle value given by the implemented controllers. The DSP board is also used to measure the joints angles and the end effector z-position of the robot arm manipulator. The parameters values of the three FSNN used to compute the three AFSNNPID controllers are given in Table 3.10.

	T_1	ω_1	N_1
First FSNN	720	0.0087	5
Second FSNN	720	0.0087	5
Third FSNN	800	0.00785	3

Table 3. 10 : Parameters values of the three FSNN used to compute the three AFSNNPID controllers.

To highlight the control performance of the implemented controllers, a comparative study of these controllers, considering various operating conditions, is carried out. In the first experiment, sinusoidal reference trajectories are used and the robot arm manipulator is free of load. Disturbances with amplitudes equal to $[30^o; 30^o; 30\ mm]$ are added to θ_1 and θ_2 and $Z - amplitude$ respectively, at the time interval [10s; 20s]. The obtained results are given in figure (3.22). For each controller and for the same initial conditions, the values of the MSE, the MAE and the RMSE are computed over the considered control interval and gathered in table (3.11). The execution time of the control algorithm is very important parameter to evaluate its computing efficiency and real time applicability. Starting the considered control algorithms from the same initial conditions, the required time to obtain a value of the control signal, using the TMS320F28335 DSP, is evaluated for each control and given in table (3.11).

	AFSNNPID	ANNPID	PSO-based PID
MSE	174.3711	181.6585	188.8117
MAE	8.5154	9.8597	9.1733
RMSE	13.2050	13.4781	13.7409
Computing time(ms)	0.7232	0.8023	3.5263

Table 3. 11 : Computing time, MSE, MAE and RMSE values for each controller.

It can be seen that a good tracking accuracy of the reference trajectories is achieved and the disturbances effect is compensated in case of the three controllers; however the AFSNNPID controller gives better control performance, than the other ones. It is clear from table (3.11) that none of the controllers exceeds the sampling time ($T_s = 10ms$), however, the AFSNNPID has the smallest computing time.

In the second experiment, the control objective is to force the robot arm to pick and drop three different loads from a given initial location to a given final location. Table (3.12) gives the weight value, the initial location and the final location of each load. The corresponding reference trajectories are generated from the cinematic model of the manipulator. Figure (3.23) shows the control performance of each controller and table (3.13) gives the corresponding MSE, MAE, RMSE and the computing time values.

	weight	Initial location (x, y)	Final location (x, y)
Load 1	0.3 Kg	$(0.2m, -0.1m)$	$(0.2m, 0.1m)$
Load 2	0.4 Kg	$(0.2m, -0.2m)$	$(0.2m, 0.2m)$
Load 3	1.4 Kg	$(0.1m, -0.15m)$	$(0.1m, 0.15m)$

Table 3. 12 : Weight values and locations of the used loads.

	AFSNNPID	ANNPID	PSO-based PID
MSE	1095.8	1125.4	1086.5
MAE	19.9691	20.6884	19.8710
RMSE	33.1022	33.5466	32.9617
Computing time(ms)	0.7276	0.7947	4.5856

Table 3. 13 : Computing time, MSE, MAE and RMSE values in case of different loads for each controller.

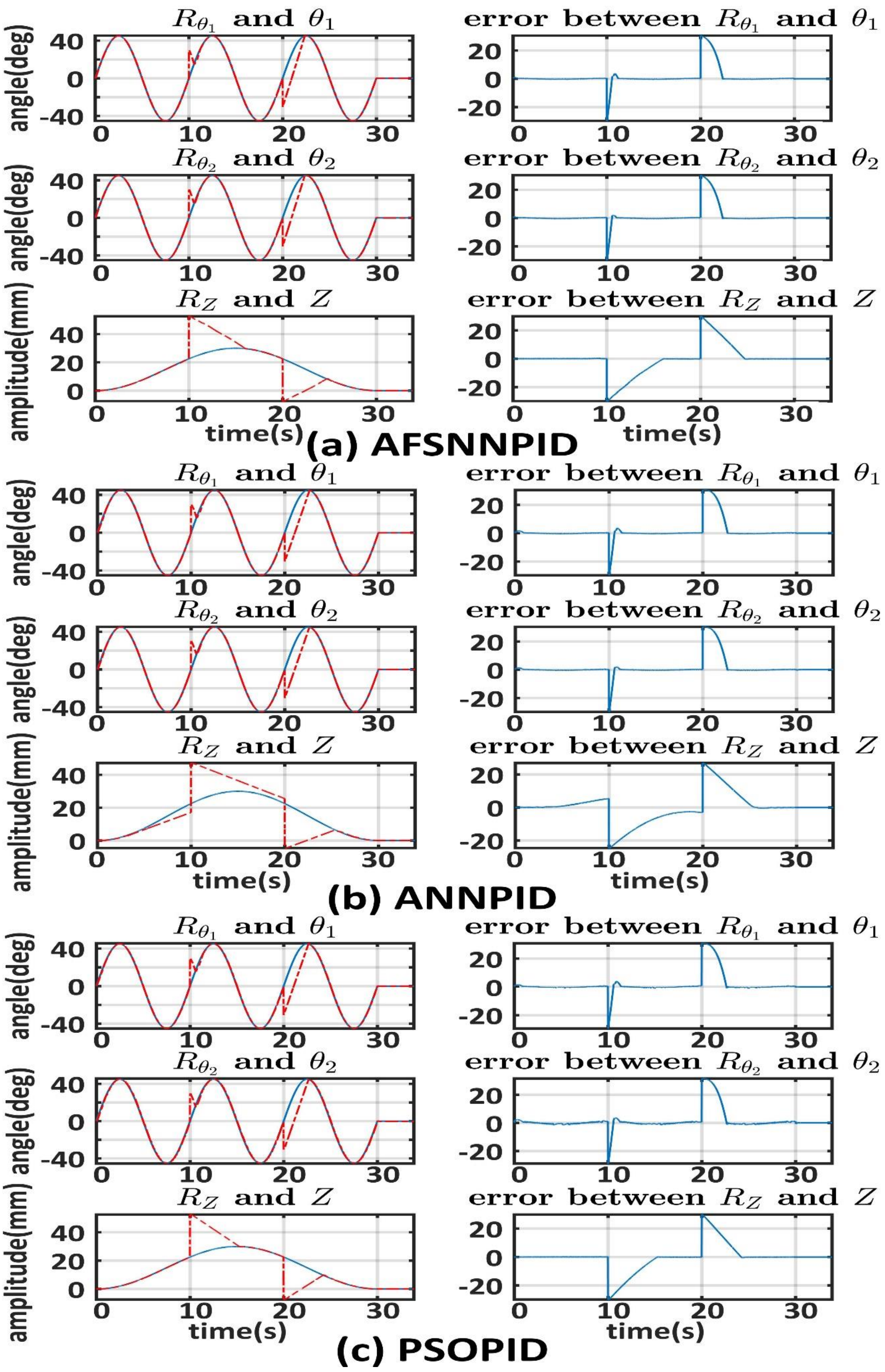


Figure 3. 22 : Control results of the free load robot arm.

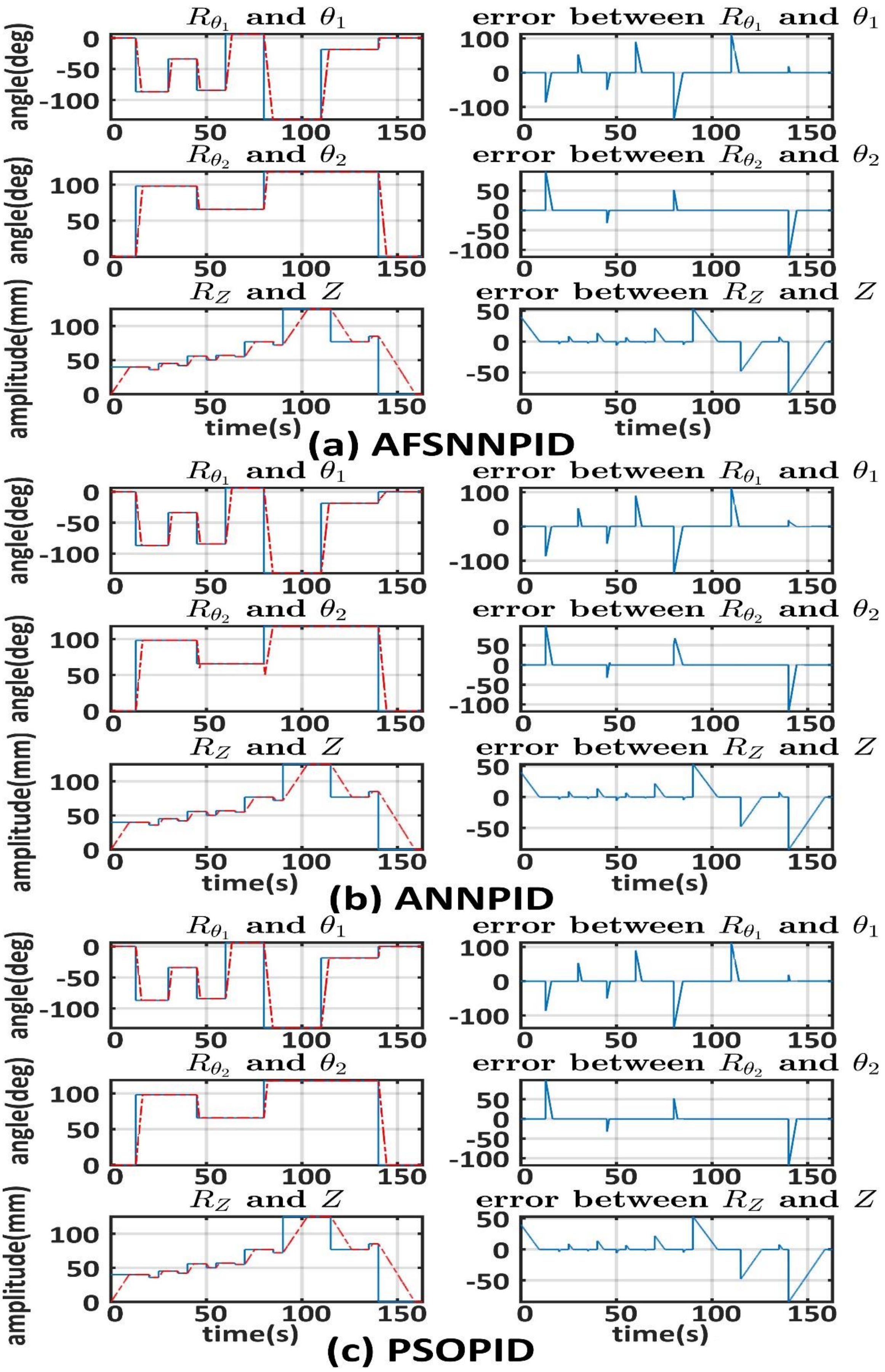


Figure 3. 23 : Control results of the robot with different loads.

From figure (3.23) and table (3.13), it can be seen that a good tracking accuracy of the corresponding reference trajectories is obtained for different load weights in case of the three controllers. Comparing the MSE, the MAE and the RMSE values, given in table (3.23), we note that the PSO based PID controller is more efficient in this experiment than the AFSNNPID and the ANNPID controllers. However, it can be seen that the AFSNNPID controller has the smallest computing time value.

In the third experiment, in order to create a gap between the system and the model outputs, the model parameters values are changed to have a gap of 5^o for both θ_1 and θ_2,and a gap of $5mm$ for the $Z - amplitude$. We have used a reference trajectory composed of four steps having a duration of 10 second and an amplitude of $[40, 0\ , -40, 0]$ degrees, for θ_1 and θ_2, and $[40, 0\ , -40, 0]$ mm for $Z - amplitude$.

The computing time, the MSE, the MAE, and the RMSE values computed over the control interval, when the model parameters values are changed, are given in table (3.14) and in table (3.15) when the model parameters values are not changed. It can be seen that the AFSNNPID controller presents the smallest values of the computing time.

From table (3.14) and (3.15), it can be observed that the difference between the values of the MSE, the MAE and the RMSE, in both cases, is small and the AFSNNPID controller gives better performance than the other controllers.

	AFSNNPID	ANNPID	PSO-based PID
MSE	609.7726	609.7847	615.3655
MAE	22.8006	22.8559	23.2492
RMSE	24.6936	24.6938	24.8066
Computing time(ms)	0.7119	0.7990	8.1327

Table 3. 14 : Computing time, MSE, MAE and RMSE values when the model parameters are changed.

	AFSNNPID	ANNPID	PSO-based PID
MSE	607.8042	608.6521	614.5759
MAE	22.7684	22.7794	23.1389
RMSE	24.6537	24.6709	24.7906
Computing time(ms)	0.7094	0.7915	6.6315

Table 3. 15 : Computing time, MSE, MAE and RMSE values when the model parameters are not changed.

7. Conclusion

In this chapter, three AI-based PID controllers, namely the ANNPID, the AFSNNPID and the PSO-based PID, have been considered. The AFSNNPID controller uses the Fourier series neural network to compute and online adjust the gains of the conventional PID controller. In fact, the Fourier series neural networks have a simple architecture and can be easily trained using the simple Delta rule. The implementation procedure of the proposed controller is simple and requires only designing two FSNN; the first one allows estimating the system Jacobian and the second is used to obtain the PID controller gains. The stability of the proposed controller has been proved using the small gain theorem and its effectiveness in controlling highly nonlinear systems has been experimentally assessed.

To assess the effectiveness of the ANNPID, the AFSNNPID and the PSO-based PID controllers, the control of the continuous stirred tank reactor and the 3-DOF robot arm manipulator, through simulation and experimental studies, has been investigated. The simulation and the experimental results have shown that these controllers give good control performance in terms of the tracking accuracy and the robustness against external disturbances and dynamic system variation. However, the proposed AFSNNPID controller do not require a large computing time, which allows it to be used in several real time applications. Indeed, the AFSNNPID controller has a simple design procedure and can be used to control any nonlinear system.

CHAPTER 4

NEURAL NETWORK MODEL PREDICTIVE CONTROL BASED ON META HEURISTIC OPTIMIZATION

1. Introduction

In this chapter, after formulating the problem of model predictive control and using neural networks and the meta heuristic optimization algorithms (TLBO, I-TLBO and ETLBO) given in the second chapter, three nonlinear model predictive control strategies are developed. The design and implementation procedure of the proposed controllers is given and their efficiency is evaluated both in simulation and experimentally.

2. Neural Network based model predictive control

2.1. Model predictive control principle

Any Model Predictive Control (MPC) strategy is based on the use of an explicit model to predict the future behavior $(\hat{y}(k+i|k),\ (i=1,\dots,N_p))$ of the controlled system over a finite prediction horizon N_p, then a cost function is optimized over a finite control horizon N_u to obtain a control sequence $(u(k+i|k),\ (i=0, N_u-1))$.

$\hat{y}(k+i|k)$ is the predicted value of the system output at the sampling time $(k+i)$ which is calculated at the sampling time k, and $u(k+i|k)$ is the control signal value at the sampling time $(k+i)$ which is calculated at the sampling time k. Usually, the control horizon is smaller than the prediction horizon $(N_u \leq N_p)$, so the control signal is taken constant beyond the control horizon $(u(k+i|k) = u(k+N_u-1|k)$ For $N_u \leq i \leq N_p)$.

After calculating the sequence of the control signal, only the first element is applied to the controlled system. The MPC strategy can be summarized by the following steps:

- Using the system model, the future values of the system outputs are calculated over the prediction horizon N_p.
- A desired reference trajectory must be specified at least over the prediction horizon N_p.
- A control sequence that minimizes a given cost function is computed. Only the first element of this sequence is applied to the controlled system.

These steps are repeated at each sampling time.

To simplify the notation, we use $\hat{y}(k+i)$ instead of $\hat{y}(k+i|k)$ to denote the output future values obtained at the sampling time k, and $u(k+i)$ instead of $u(k+i|k)$ to denote the control future values computed at the sampling time k.

2.1.1. Cost function

The cost function includes all the desired control objectives over the prediction and the control horizons. The common used cost function is given by following quadratic form:

- The error between the desired reference trajectories and the predicted system outputs $\hat{Y}(k)$.
- The control signal increment ($\Delta u(k)$).

$$J\big(\Delta u(k), \hat{y}(k), c(k)\big) = \sum_{i=N_1}^{N_2} \left[\big(\hat{y}(k+i/k) - c(k+i)\big)^T Q\big(\hat{y}(k+i/k) - c(k+i)\big)\right] + \sum_{i=1}^{N_u} [\Delta u(k+i-1)^T R\, \Delta u(k+i-1)] \tag{4.1}$$

where, $C(k)$ is the desired reference trajectory, $\Delta u(k) = u(k) - u(k-1)$ is the control increment, and N_1 and N_2 are the lower and the upper limits of the prediction horizon ($N_p = N_2 - N_1 + 1$), respectively. The weight matrices Q and R are semi positive defined and positive defined matrices, respectively.

Generally, the value of N_1 is chosen according to the delay time of the controlled system and the control horizon N_u must not exceed N_2 ($1 \le N_u \le N_2$). In some cases, to reduce the complexity of the optimization problem, the weight matrix R is chosen to be a NULL matrix, and the weight matrix Q is chosen to be an identity matrix. Therefore, the optimization problem, given by equation (4.1), become:

$$J\big(\Delta u(k), \hat{y}(k), c(k)\big) = \sum_{i=N_1}^{N_2} \left\|\big(\hat{y}(k+i/k) - c(k+i)\big)\right\|^2 \tag{4.2}$$

2.1.2. <u>Constraints</u>

One of the most important strengths of the MPC technique is its constraints handling capabilities. In fact, the majority of physical systems have some limitations imposed on their variables, which have to be included in the optimization problem in the form of different constraints. Most of these constraints can be characterized as follows:

- Constraints on the inputs:

$$u_{min} \le u(k+i) \le u_{max} \qquad i = 0,1,\dots,N_u - 1 \tag{4.3}$$

- Constraints on the inputs increments:

$$\Delta u_{min} \le \Delta u(k+i) \le \Delta u_{max} \qquad i = 0,1,\dots,N_u\text{-}1 \tag{4.4}$$

- Constraints on the outputs:

$$y_{min} \le \hat{y}(k+i) \le y_{max} \qquad i = N_1, \dots, N_2 \tag{4.5}$$

Other types of constraints could be considered, such as: constraints on the outputs increments, constraints on the state variables of the controlled system, constraints on the increments of the state variables … etc.

By considering the source of their origin, the constraints can be classified in two categories:

- Hard constraints: they are imposed by physical limitations of the controlled system. Therefore, they cannot be exceeded. In case of violation of any of these constraints, a physical damage in the controlled system is occurred immediately. Inputs constraints are usually considered as hard constraints, the origins of these constraints are the working limit ranges of the actuators.

- Soft constraints: these constraints are imposed by technological nature, safety, economic and environmental objectives. Hence, they can be physically exceeded. However, this violation must be temporary and under special conditions. Outputs constraints are usually considered as soft constraints.

By tacking in account the imposed constraints, the optimization problem of MPC is given as follows:

$$J\big(\Delta u(k), \hat{y}(k), c(k)\big) = \sum_{i=N_1}^{N_2} \left[\big(\hat{y}(k+i/k) - c(k+i)\big)^T Q \,\big(\hat{y}(k+i/k) - c(k+i)\big)\right] + \sum_{i=1}^{N_u} [\Delta u(k+i-1)^T \, R \, \Delta u(k+i-1)] \tag{4.6}$$

Subject to:

$$\begin{gathered} \Delta u(k+i-1) = 0 \text{ for } i > N_u \\ y_{min} < \hat{y}(k+i) < y_{max} \text{ for } \; i = N_1, \dots, N_2 \\ u_{min} < u(k+i) < u_{max} \text{ for } \; i = 0,1, \dots, N_u - 1 \\ \Delta u_{min} < \Delta u(k+i) < \Delta u_{max} \text{ for } i = 0,1, \dots, N_u - 1 \end{gathered}$$

2.1.3. Prediction model

As mentioned above, the future outputs of the controlled system are predicted using a suitable model of the system. Therefore, the first step in designing any MPC algorithm is obtaining a model that have the ability to mimic the dynamics of the controlled system with negligible error. Three different approaches that can be used to design the prediction model are presented as follows:

- The black box approach: The input/output database is required to build such models. The prior knowledge about the system dynamics are not required.
- The white box approach: The input/output database is not needed; however, the system's balance equations must be known to build this model. The white box model gives better performance than the black box model; however, it is difficult to build such models.
- The gray box approach: this approach is a hybrid between the black box and the white box approaches, it requires some of the system balance equations and an input/output database. This approach gives good modeling performance and it is less complex than the white box approach.

Several models can be used to predict the future behavior of the controlled system. However, the chosen prediction model should be the simplest one that can give sufficiently precise predictions. In case of a linear prediction model, and no imposed constraints, the optimization problem becomes a quadratic function, which has a unique global minimum; therefore, an analytical solution can be obtained. However, the most physical systems are nonlinear and subjected to different inputs and outputs constraints. To obtain acceptable performance, nonlinear models should be used and the different constraints must be incorporated in the optimization problem. In this case, analytical solutions, to the non-convex and nonlinear optimization problem, do not exist and numerical optimization methods should be used.

2.2. <u>Nonlinear model predictive control</u>

Although, linear MPC techniques give good control performance in many practical applications [87–89] , in the case of highly nonlinear systems, severe degradations in the control performance can be observed. Therefore, to ensure good control performance, nonlinear MPC (NMPC) methods that use a nonlinear prediction model should be investigated. In fact, a lot of attention was given to the NMPC techniques, and several control algorithms were proposed [90–92,140]. The main difficulties in designing any NMPC algorithm are obtaining an adequate nonlinear model for the system to be controlled and online solving the non-convex and nonlinear optimization problem. Obviously, the efficiency and computational requirement of the controller depend extremely on the accuracy and simplicity of the used model. Actually, there is no clearly suitable modeling approach to represent general nonlinear systems. Hence, several nonlinear models were developed and used in predictive control, such as: Volterra series [93–95], neural network models [88,90,100], fuzzy logic models [91,97,98], fuzzy neural network models [148–150],… etc.

Using a nonlinear prediction model implies a non-convex and nonlinear optimization problem, which requires a complex and time-consuming optimization algorithm to find a solution for the optimization problem. The objective of most NMPC techniques is to find a suboptimal solution that satisfies the desired control performance.

To solve the predictive control problem, the following approaches can be envisaged:

- NMPC using successive linearization: the goal of this method is to use a prediction model that gives better performance than that of a linear model and maintain the quadratic form of the optimization problem. Therefore, at each sampling time, the prediction model is linearized around the current operating conditions. After that, the linearized model is used with any linear MPC strategy [81,83,84,229,230]. Due to the approximation of the prediction model with a linear one, this strategy is considered as a suboptimal approach. The NMPC with successive linearization gives good control performance if the controlled system have slow dynamics, in this case, it is not required to perform the linearization at each sampling time, but rather after a given number of samples, this number depend on the dynamics rate of the controlled system. It is clear that, in the case of systems with fast dynamics, such approach could be insufficient.
- NMPC using nonlinear predictions and linearization: obviously, the previous approach is limited especially when it comes to control nonlinear systems with fast dynamics. Hence, to enhance the control performance, in this approach, the superposition principle is used, where the system response is decomposed into free and forced responses. The aim of this decomposition is to facilitate solving the optimization problem. Therefore, a nonlinear prediction model is used to evaluate the free response while a linearized one is used to evaluate the forced response. The resulting optimization problem is quadratic and convex as in the precedent approach. But, due to using a nonlinear model to calculate the free

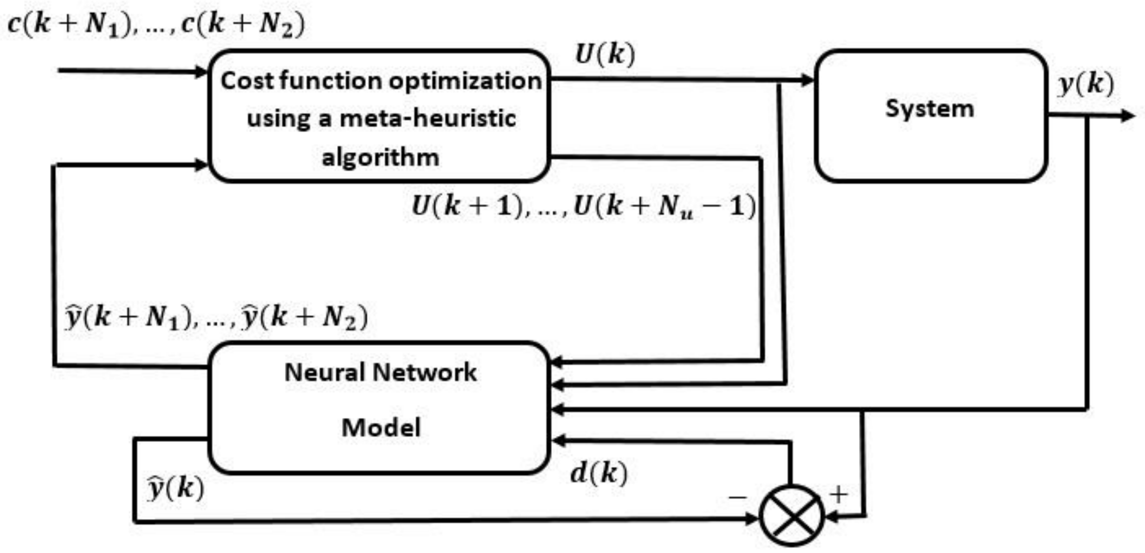
response, this approach gives better control performance than the precedent approach.

- NMPC using nonlinear optimization method: in this approach, a nonlinear prediction model is used to generate the response of the controlled system, and the associated optimization problem is nonlinear and non convex. The optimization methods that can be used in this case can be classified into the following two main families:
    - Deterministic numerical methods: these methods gives a numerical The following methods can be used to solve the optimal control problem:
        - ✓ Hamilton-Jacobi-Bellman partial differential equation [231,232].
        - ✓ Euler-Lagrange differential equation [233,234].
        - ✓ Direct methods [235].
    - Stochastic numerical methods: the aim of these methods is to solve the non-quadratic and nonlinear optimization problem using stochastic meta-heuristic algorithms. These algorithms are based on fundamental elements that produce evolutionary intelligent behavior in natural systems. Meta heuristic algorithms are known of their ability to handle the most optimization problem; they have good performance, and could locate adequate solutions in a reasonable time. In fact, many research works have used meta heuristic optimization methods to solve the NMPC optimization problem. Particularly, the following methods have been used:
        - ✓ Genetic algorithm [104,105].
        - ✓ Particle swarm optimization algorithm [90,106,107].
        - ✓ Artificial bee colony [96,97].
        - ✓ Evolutionary algorithm [108].

2.3. <u>Neural network based model predictive control</u>

Neural networks are capable of approximating any given function with arbitrary precision [236]; they are universal approximators. Due to their simple structure and good precision, neural networks are very suitable for NMPC. Several architectures of neural networks can be found in the literature. Such as: the feed forward neural networks, the recurrent neural networks, the radial basis neural networks and the Elman network. Each one of these architectures has its own properties and can be used to build the prediction model used in NNMPC. To obtain such models, a training step of the network weights is required. Several training algorithms have been proposed, such as: the quasi-Newton method [237], the Levenberg-Marquardt algorithm [238] and the famous back propagation algorithm [239]. In the present thesis, the Multi-Layer Perceptron (MLP) with one hidden layer and the back propagation algorithm are used.

Assuming that, the controlled system is a MIMO process which has m inputs and n outputs, and a MLP NN is used as the prediction model, the control block diagram is given by figure (4.1).



$c(k+N_1),\dots,c(k+N_2)$
Cost function optimization using a meta-heuristic algorithm
$U(k)$
System
$y(k)$
$U(k+1),\dots,U(k+N_u-1)$
$\hat{y}(k+N_1),\dots,\hat{y}(k+N_2)$
Neural Network Model
$\hat{y}(k)$
$d(k)$
$-$
$+$

Figure 4. 1 : Control block diagram of the NNMPC.

2.4. Solving the NNMPC optimization problem

The NNMPC optimization problem, generally defined by equation (4.6), must be solved to obtain the desired control action.

2.4.1. Constraint handling

One of the interesting advantages of predictive control is its ability to efficiently and directly handle the constraints by incorporating theme in the formulation of the optimization problem. In the present work, the input constraints are considered as hard constraints. They are directly handled by bounding the search space using a preserving strategy. The output constraints are considered as soft constraints; they are handled using the penalizing approach.

2.4.1.1. Output constraints

As cited above, the penalizing approach is used to handle the output constraint. In this approach, to heavily penalize any constraints violation, new variables, called slack variables, are added to the cost function. Using this approach, the optimization problem given by equation (4.6) is reformulated to have the following expression:

$$J\big(\Delta u(k),\hat{y}(k),c(k)\big)=\sum_{i=N_1}^{N_2}\Big[\big(\hat{y}(k+i/k)-c(k+i)\big)^T\Gamma_y\big(\hat{y}(k+i/k)-c(k+i)\big)\Big] + \sum_{i=1}^{N_u}[\Delta u(k+i-1)^T R\,\Delta u(k+i-1)] \tag{4.7}$$

Subjected to:

$$\Delta u(k+i-1)=0 \text{ for } i>N_u$$
$$u_{min}<u(k+i)<u_{max} \text{ for } i=0,1,,\dots,N_u-1$$
$$\Delta u_{min}<\Delta u(k+i)<\Delta u_{max} \text{ for } i=0,1,,\dots,N_u-1$$

where, the output-dependent weight function $\Gamma_y(y)$ is chosen to replace the imposed output constraint and has the following expression:

$$\Gamma_y(\hat{y}) = \begin{pmatrix} \Gamma_{\hat{y}_1}(\hat{y}_1) & 0 & \dots & 0 \\ 0 & \Gamma_{\hat{y}_2}(\hat{y}_2) & \dots & 0 \\ 0 & \vdots & \ddots & 0 \\ 0 & \cdots & 0 & \Gamma_{\hat{y}_q}(\hat{y}_q) \end{pmatrix} \tag{4.8}$$

such as:

$$\Gamma_{\hat{y}_i}(\hat{y}_i) = \begin{cases} \Gamma_{\hat{y}_i}(0) & if\ y_{min_i} \le \hat{y}_i \le y_{max_i} \\ \Gamma_{\hat{y}_i}(0)\left[1 + C_{i_y}\right] & otherwise \end{cases}$$

where, $i = 1, \dots, q$ (q is the number of outputs) and C_{i_y} is used to define the degree of penalization: $C_{i_y} = 0$ indicates no constraint, while $C_{i_y} = \infty$ indicates hard constraint.

2.4.1.2. <u>Input constraints</u>

In population-based algorithms the input constraints can be systematically handled by bounding the search space to the inputs admissible values. The inputs constraints to be handled are:

- Constraints on the input increment.

$$\Delta u_{min} < \Delta u(k) < \Delta u_{max}$$

- Constraints on the input magnitude.

$$u_{min} < u(k) < u_{max}$$

These constraints can be combined into one single constraint as follows:

$$\begin{aligned} &\Delta u_{min} < \Delta u(k) < \Delta u_{max} \\ &\to \Delta u_{min} + u(k-1) < \Delta u(k) + u(k-1) < \Delta u_{max} + u(k-1) \\ &\to \Delta u_{min} + u(k-1) < u(k) < \Delta u_{max} + u(k-1) \end{aligned} \tag{4.9}$$

According to the constraint on the input magnitude and equation (4.9), the upper and the lower bounds of the input magnitude will be given by:

$$L_{min}(k) < u(k) < L_{\max}(k) \tag{4.10}$$

where:

$$L_{min}(k) = \begin{cases} u_{min} & if\ \ u_{min} > \Delta u_{min} + u(k-1) \\ \Delta u_{min} + u(k-1) & otherwise \end{cases}$$

$$L_{max}(k) = \begin{cases} u_{max} & if\ \ u_{max} < \Delta u_{max} + u(k-1) \\ \Delta u_{max} + u(k-1) & otherwise \end{cases}$$

Equation (4.10) gives the upper and lower limits of the search space. Finally the NMPC optimization problem becomes:

$$J(\Delta u(k), \hat{y}(k), c(k)) = \sum_{i=N_1}^{N_2} \left[(\hat{y}(k+i/k) - c(k+i))^T \Gamma_y (\hat{y}(k+i/k) - c(k+i)) \right] + \sum_{i=1}^{N_u} [\Delta u^T(k+i-1) R \Delta u(k+i-1)] \tag{4.11}$$

Subjected to:

$$\Delta u(k+i-1) = 0 \text{ for } i > N_u$$
$$L_{min}(k) < u(k) < L_{\max}(k)$$

3. Proposed control algorithms

Assuming that the neural network prediction model is obtained and the parameters of the NNMPC $(N_u, N_1, N_2, \varepsilon, C_{i_y})$ are chosen, the proposed control algorithms are detailed in the following subsections.

3.1. TLBO based NNMPC algorithm

In this algorithm, the basic teaching learning based optimization algorithm is used to solve the constrained nonlinear optimization problem given by equation (4.11). Assuming that the TLBO parameters (m, n and k_{max}) are established, the steps of this control strategy (NNMPC-TLBO) are given as follows:

Step 1: Initialization

- Let us take the control inputs at the sampling time k for the i^{th} iteration as $X_{j\,k}^{i}$, $(j = 1, \ldots, m)$, where m denotes the number of control inputs.
- For j=1: m
    - For k=1: n
        - Choose the initial solution $X_{j\,k}^{1}$ using equation (2.15).
    - End
- End
- i=1.

Step 2: Reference trajectory

- Specify the reference trajectory between $\mathrm{k} + N_1$ and $\mathrm{k} + N_2$.

Step 3: Teacher phase

- **Step 3_1:** Determination of the teacher
    - For k=1: n
        - Calculate the predicted values of the system outputs using the prediction model.
        - Evaluate the objective function F_k^i using equation (4.11).
    - End
    - $F_{k_{best}} = F_1^i$
    - For k=2: n
        - If F_k^i<$F_{k_{best}}$

```
            ✓ $F_{k_{best}} = F_k^i$
            ✓ $X_{jkbest} = X_{jk}^i$
        ➢ End if
    o End
• Step 3_2: Mean result calculation
    o For $j$=1: $m$
        ➢ Calculate the mean result $M_j^i$ using equation (2.10).
    o End
• Step 3_3: difference mean calculation
    o For $j$=1: $m$
        ➢ For $k$=1: $n$
            ✓ Calculate the difference mean ($d_{j\,k}^i$) using equation (2.11).
        ➢ End
    o End
• Step 3_4: Solution updating
    o For $j$=1: $m$
        ➢ For $k$=1: $n$
            ✓ Calculate the new solutions $Xnew_{j\,k}^i$using equation (2.12).
        ➢ End
    o End
• Step 3_5: Greedy selection
    o For $k$=1: $n$
        ➢ Using $Xnew_{j\,k}^i$ , calculate the predicted values of the system outputs using
          the prediction model.
        ➢ Evaluate the objective function $new\_F_k^i$ using equation (4.11).
        ➢ If $new\_F_k^i$ < $F_k^i$
            ✓ $F_k^i = new_{F_k^i}$
            ✓ $F_{j\,k}^i = Xnew_{j\,k}^i$
        ➢ End if
        ➢ If $F_k^i$<$F_{k_{best}}$
            ✓ $F_{k_{best}} = F_k^i$
            ✓ $X_{jkbest} = X_{j\,k}^i$
        ➢ End if
    o End
Step 4: Learner phase
• Step 4_1: choosing the pairs to interact
    o Choose randomly $q$ pairs of solutions such that $F_A^i \neq F_B^i$ where $F_A^i$ and $F_B^i$ are the
      objective function values of $X_A$ and $X_B$ respectively.
• Step 4_2: Solutions updating
    o For $h$=1: $q$
        ➢ Update the solution $Xnew_{j\,k}^i$ using equations (2.13) and (2.14).
    o End
• Step 4_3: evaluating the new solutions
```

```
    o For $k$=1: $n$
        ➢ Using $Xnew^{i}_{j\,k}$, calculate the predicted values of the system outputs using the prediction model.
        ➢ Evaluate the objective function $new\_F^{i}_{k}$ using equation (4.11).
    o End
    o For $k$=1: $n$
        ➢ If $new\_F^{i}_{k} < F^{i}_{k}$
            ✓ $F^{i}_{k} = new\_F^{i}_{k}$
            ✓ $X^{i}_{j\,k} = Xnew^{i}_{jk}$
        ➢ End if
        ➢ If $F^{i}_{k} < F_{k_{best}}$
            ✓ $F_{k_{best}} = F^{i}_{k}$
            ✓ $X_{j\,kbest} = X^{i}_{j\,k}$
        ➢ End if
    o End
```

Step 5: iterative process

```
• if $F_{k_{best}} < \varepsilon \;\; or \; i > k_{max}$
    o $i$=1.
    o Go to step 6.
• Else
    o $i = i + 1$.
    o Go back to step 3.
• End
```

Step 6:

```
• Apply the obtained control value (the first element of $X_{j\,kbest}$) on the system.
• Wait for the next sampling time, and then go back to step 2.
```

3.2. <u>I-TLBO based NNMPC algorithm</u>

In this algorithm, the improved teaching learning based optimization algorithm is used to solve the constrained nonlinear optimization problem given by equation (4.11). Assuming that the I-TLBO parameters ($T_n, m,\ n$ and k_{max}) are established, the basic steps of the NNMPC-ITLBO are given as follows:

Step 1: Initialization

```
• Let us take the control inputs at the sampling time k for the $i^{th}$ iteration as $X^{i}_{j\,k}$, ($j = 1, \dots, m$), where $m$ denotes the number of control inputs.
• For $j$=1: $m$
    o For $k$=1: $n$
        ➢ Choose the initial solution $X^{1}_{j\,k}$ using equation (2.15).
    o End
• End
• $i$=1.
```

Step 2: Reference trajectory

```
• Specify the reference trajectory between $k + N_1$ and $k + N_2$.
```

Step 3: Determination of the first teacher

```
• For $k$=1: $n$
    o Calculate the predicted values of the system outputs using the prediction model.
    o Evaluate the objective function $F_k^i$ using equation (4.11).
• End
• $F_{(X_{teacher})_1} = F_1^i$
• For $k$=2: n
    o If $F_k^i$<$F_{(X_{teacher})_1}$
        ➢ $F_{(X_{teacher})_1} = F_k^i$
        ➢ $(X_{teacher})_1 = X_{jk}^i$
    o End if
• End
```

Step 4: Determination of the other $(T_n - 1)$ teachers

```
• Select the other teachers using equation (2.16)
```

Step 5: assigning groups to the teachers

```
• For $k = 1: (n - T_n)$
    o If $F_{(X_{teacher})_1} \geq F_k^i > F_{(X_{teacher})_2}$
        ➢ Assign the learner $X_k^i$ to the teacher $(X_{teacher})_1$
    o Else, If $F_{(X_{teacher})_2} \geq F_k^i > F_{(X_{teacher})_3}$
        ➢ Assign the learner $X_k^i$ to the teacher $(X_{teacher})_2$

                                  ⋮

    o Else, If $F_{(X_{teacher})_{T_n-1}} \geq F_k^i > F_{(X_{teacher})_{T_n}}$
        ➢ Assign the learner $X_k^i$ to the teacher $(X_{teacher})_{T_n-1}$
    o Else
        ➢ Assign the learner $X_k^i$ to the teacher $(X_{teacher})_{T_n}$
• End
```

Step 6: Mean result calculation for each group

```
• For $i = 1: T_n$
    o For $j$=1: $m$
        ➢ Calculate the mean result $M_j^i$ for each group using equation (2.10).
    o End
• End
```

Step 7: Adaptive teaching factor and difference mean calculation

```
• For $i = 1: T_n$
    o Calculate the Adaptive teaching factor using equation (2.17).
    o For $j$=1: $m$
        ➢ For $k$=1: $n$
            ✓ Calculate the difference mean $(d_{j\,k}^{i})$ for each group.
        ➢ End
    o End
• End
```

```
Step 8: Learning through tutorial hours
  • For $i = 1{:}T_n$
      o For $j$=1: $m$
          ➢ For $k$=1: $n$
              ✓ Calculate the new solutions $Xnew_{j\,k}^{i}$ using equations (2.18) and (2.19).
          ➢ End
      o End
  • End
Step 9: Greedy selection
  • For $k$=1: $n$
      o Using $Xnew_{j\,k}^{i}$ , calculate the predicted values of the system outputs using the
        prediction model.
      o Evaluate the objective function $new\_F_k^i$ using equation (4.11).
      o If $new\_F_k^i < F_k^i$
          ➢ $F_k^i = new_{F_k^i}$
          ➢ $F_{j\,k}^{i} = Xnew_{j\,k}^{i}$
      o End if
      o If $F_k^i$<$F_{k_{best}}$
          ➢ $F_{k_{best}} = F_k^i$
          ➢ $X_{jkbest} = X_{j\,k}^{i}$
      o End if
  • End

Step 10: choosing the pairs to interact
  • Choose randomly $q$ pairs of solutions such that $F_A^i \neq F_B^i$ where $F_A^i$ and $F_B^i$ are the
    objective function values of $X_A$ and $X_B$ respectively.
Step 11: Self-motivated learning
  • For $h$=1: $q$
      o Update the solution $Xnew_{j\,k}^{i}$ using equations (2.20) and (2.21).
  • End
Step 12: evaluating the new solutions
  • For $k$=1: $n$
      o Using $Xnew_{j\,k}^{i}$, calculate the predicted values of the system outputs using the
        prediction model.
      o Evaluate the objective function $new\_F_k^i$ using equation (4.11).
  • End
  • For $k$=1: $n$
      o If $new\_F_k^i < F_k^i$
          ➢ $F_k^i = new\_F_k^i$
          ➢ $X_{j\,k}^{i} = Xnew_{jk}^{i}$
      o End if
      o If $F_k^i$<$F_{k_{best}}$
          ➢ $F_{k_{best}} = F_k^i$
        ➢ $X_{j\,kbest} = X_{j\,k}^{i}$
    o End if
• End
```

Step 13: iterative process

```
• if $F_{k_{best}} < \varepsilon \;\; or\; i > k_{max}$
    o $i$=1.
    o Go to step 14.
• Else
    o $i = i + 1$.
    o Go back to step 3.
• End
```

Step 14:

- Apply the obtained control value (the first element of $(X_{teacher})_1$) on the system.
- Wait for the next sampling time, and then go back to step 2.

3.3. ETLBO based NNMPC Algorithm

In this algorithm, the proposed enhanced teaching learning based optimization algorithm is used to solve the constrained nonlinear optimization problem given by equation (4.11). Assuming that the ETLBO parameters (m, n and k_{max}) are established, the proposed NNMPC-ETLBO algorithm is described by the flow chart given in figure (4.2).

4. Control of the 2-DOF robot arm manipulator

4.1. System presentation

To evaluate the performance of the proposed controller, the control of the model of the 2-DOF robot arm manipulator, presented in [240], is considered. This planar robotic manipulator is given by figure (4.3). The dynamic model of the manipulator is expressed as follows:

$$\begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{pmatrix}\begin{pmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{pmatrix} + \begin{pmatrix} P_{11} \\ P_{21} \end{pmatrix} + \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} + \begin{pmatrix} g_{1q} \\ g_{2q} \end{pmatrix} = \begin{pmatrix} \tau_{f_{in1}} \\ \tau_{f_{in2}} \end{pmatrix} \tag{4.12}$$

where:

$\tau_{f_{in1}}, \tau_{f_{in2}}$ are the control torques for joints 1 and 2, respectively,

$Q_{11} = I_1 + I_2 + m_1 l_{c1}^2 + m_2 l_1^2 + m_2 l_{c2}^2 + 2 m_2 l_1 l_{c2} cos\theta_2 + m_{33}(l_1^2 + l_2^2 + 2 l_1 l_2 cos\theta_2)$.

$Q_{12} = Q_{21} = I_2 + m_2 l_{c2}^2 + m_2 l_1 l_{c2} cos\theta_2 + m_{33}(l_2^2 + l_1 l_2 cos\theta_2)$.

$Q_{22} = I_2 + m_2 l_{c2}^2 + m_{33} l_2^2$.

$P_{11} = -l_1(2\dot{\theta}_1 + \dot{\theta}_2)\dot{\theta}_2 sin\theta_2 (l_{c2} m_2 + l_2 m_{33})$.

$P_{21} = l_1 l_{c2} \dot{\theta}_1^2 sin\theta_2 (m_2 + m_{33})$.

$f_1 = b_{1q}\dot{\theta}_1$. $f_2 = b_{2q}\dot{\theta}_2$.

$g_{1q} = m_1 l_{c1} g cos\theta_1 + m_2 g (l_{c2}\cos(\theta_1 + \theta_2) + l_1 cos\theta_1) + m_{33} g (l_2 \cos(\theta_1 + \theta_2) + l_1 cos\theta_1)$.

$g_{2q} = (m_2 + m_{33}) g l_{c2} \cos(\theta_1 + \theta_2)$.

The values of the different constants are given in table (4.1).

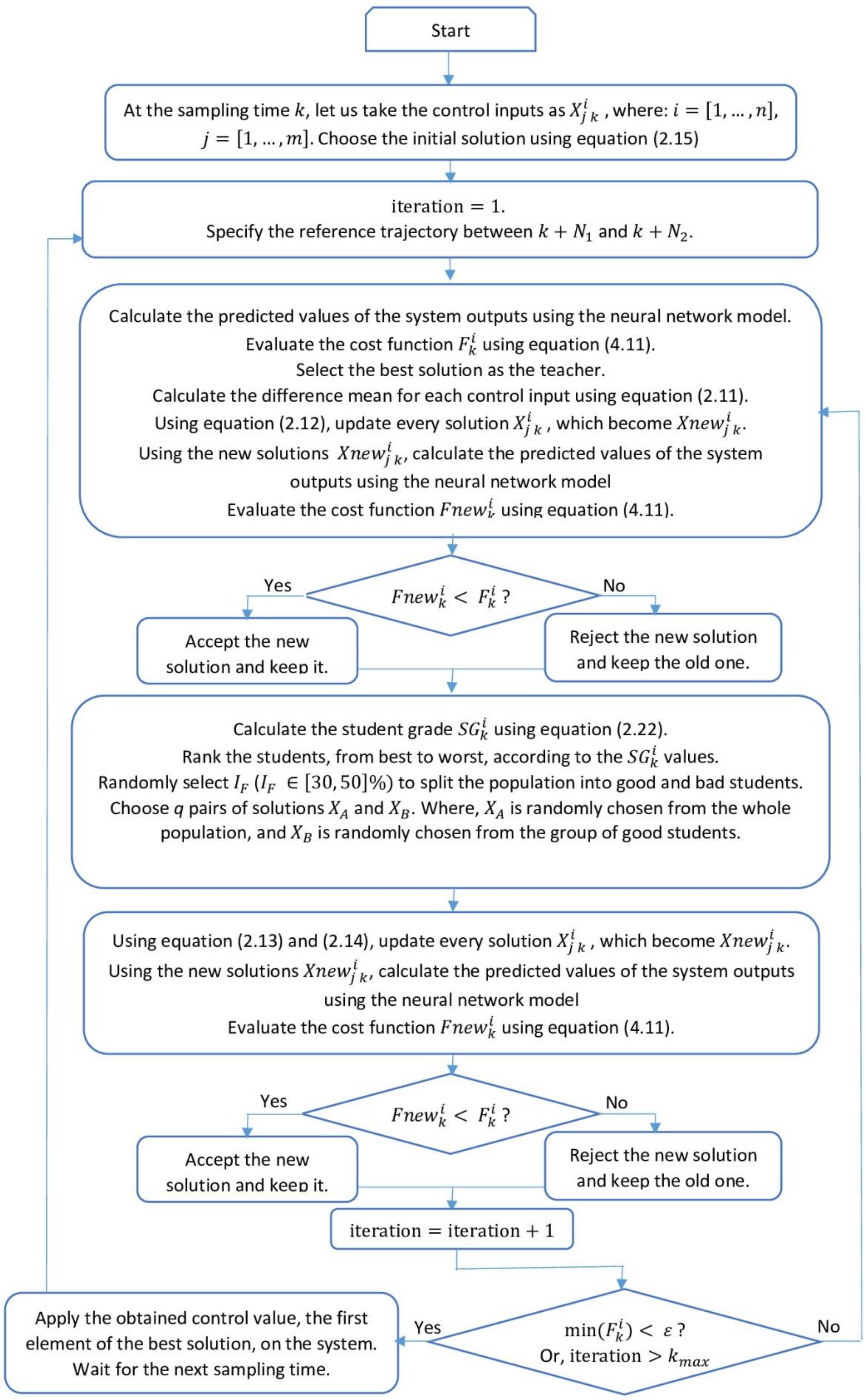

Start
At the sampling time k, let us take the control inputs as $X_{j\,k}^{i}$, where: $i = [1, \dots, n]$, $j = [1, \dots, m]$. Choose the initial solution using equation (2.15)
$\text{iteration} = 1.$
Specify the reference trajectory between $k + N_1$ and $k + N_2$.
Calculate the predicted values of the system outputs using the neural network model.
Evaluate the cost function F_k^i using equation (4.11).
Select the best solution as the teacher.
Calculate the difference mean for each control input using equation (2.11).
Using equation (2.12), update every solution $X_{j\,k}^{i}$, which become $Xnew_{j\,k}^{i}$.
Using the new solutions $Xnew_{j\,k}^{i}$, calculate the predicted values of the system outputs using the neural network model
Evaluate the cost function $Fnew_k^i$ using equation (4.11).
Yes
$Fnew_k^i < F_k^i$?
No
Accept the new solution and keep it.
Reject the new solution and keep the old one.
Calculate the student grade SG_k^i using equation (2.22).
Rank the students, from best to worst, according to the SG_k^i values.
Randomly select I_F ($I_F \in [30, 50]\%$) to split the population into good and bad students.
Choose q pairs of solutions X_A and X_B. Where, X_A is randomly chosen from the whole population, and X_B is randomly chosen from the group of good students.
Using equation (2.13) and (2.14), update every solution $X_{j\,k}^{i}$, which become $Xnew_{j\,k}^{i}$.
Using the new solutions $Xnew_{j\,k}^{i}$, calculate the predicted values of the system outputs using the neural network model
Evaluate the cost function $Fnew_k^i$ using equation (4.11).
Yes
$Fnew_k^i < F_k^i$?
No
Accept the new solution and keep it.
Reject the new solution and keep the old one.
$\text{iteration} = \text{iteration} + 1$
$\min(F_k^i) < \varepsilon$?
Or, $\text{iteration} > k_{max}$
Yes
No
Apply the obtained control value, the first element of the best solution, on the system. Wait for the next sampling time.

Figure 4. 2 : Flow chart of the proposed NNMPC-ETLBO algorithm.

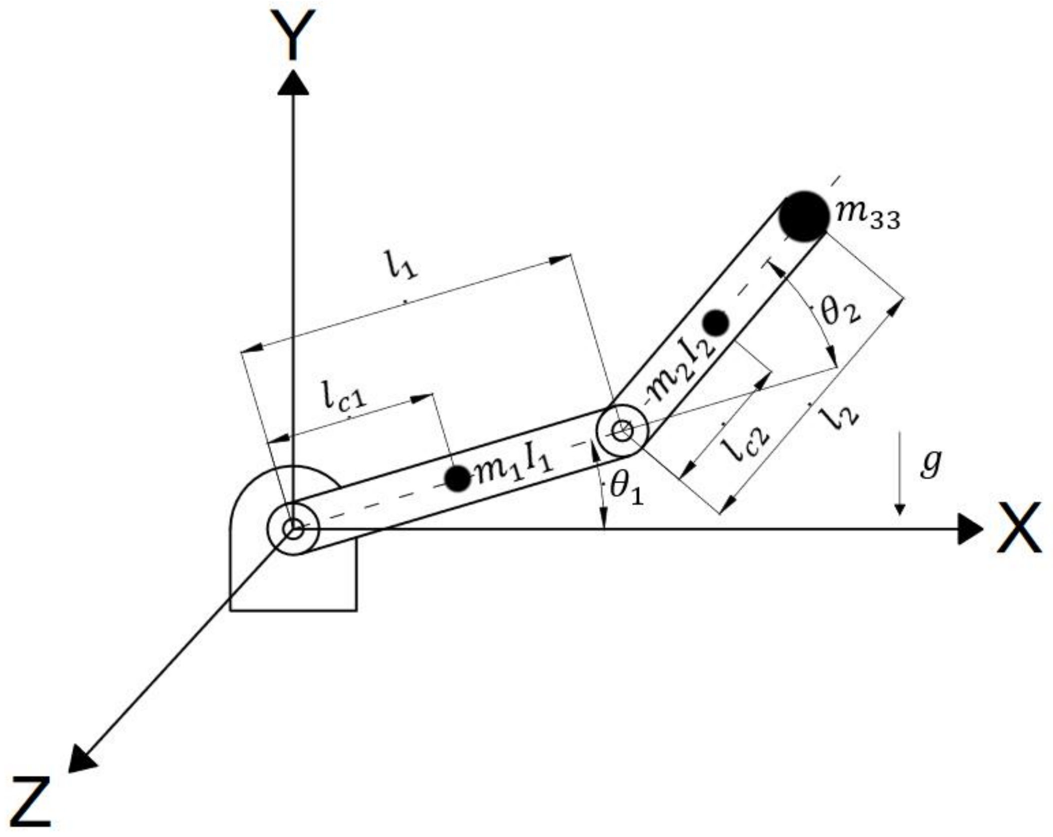


Figure 4. 3 : Planar robot arm manipulator.

parameter	value	parameter	value
m_1	0.392924 (kg)	l_{c1}	0.104648 (m)
m_2	0.094403 (kg)	l_{c2}	0.081788 (m)
m_{33}	0.2 (kg)	I_1	0.0011411 $(\text{kg} \cdot \text{m}^2)$
g	9.81 (m/s^2)	I_2	0.0020247 $(\text{kg} \cdot \text{m}^2)$
l_1	0.2032 (m)	b_{1q}	0.141231 (N)
l_2	0.1524 (m)	b_{2q}	0.3530776 (N)

Table 4. 1 : Parameters values of the considered manipulator.

4.2. Neural network modeling of the robot arm manipulator

The prediction model consists of two MLP neural networks with the following architecture:

- For both MLPs, the inputs layer contains 8 neurons, and the inputs vector is defined by:
  $\left[\tau_{f_{in1}}(k), \tau_{f_{in1}}(k-1), \tau_{f_{in2}}(k), \tau_{f_{in2}}(k-1), \theta_1(k), \theta_1(k-1), \theta_2(k), \theta_2(k-1)\right]$.
- For both MLPs, one hidden layer containing 20 neurons with a sigmoid activation functions, is used.
- The output layer contains one neuron with a linear activation function for each MLP. The output of the first MLP gives the estimated value $\hat{\theta}_1(k)$ of θ_1, and the output of the second MLP gives the estimated value $\hat{\theta}_2(k)$ of θ_2.

Using the state model, given by equation (4.12), a dataset is generated using random values of the system inputs $(\tau_{f_{in1}}, \tau_{f_{in2}})$. The generated dataset is divided into two subsets to train and test the neural network models. Figure (4.4) shows a part of the test results of the obtained models; it can be seen that the modelling error is quite small. The values of the RMSE, and the R^2, for the model of θ_1 and the model of θ_2 are: 0.00200654, 0.99999950718, 0.00211524 and 0.99999277711, respectively. The values of R^2 are close to 1 and the RMSE values are close to 0, hence the obtained models have good accuracy.

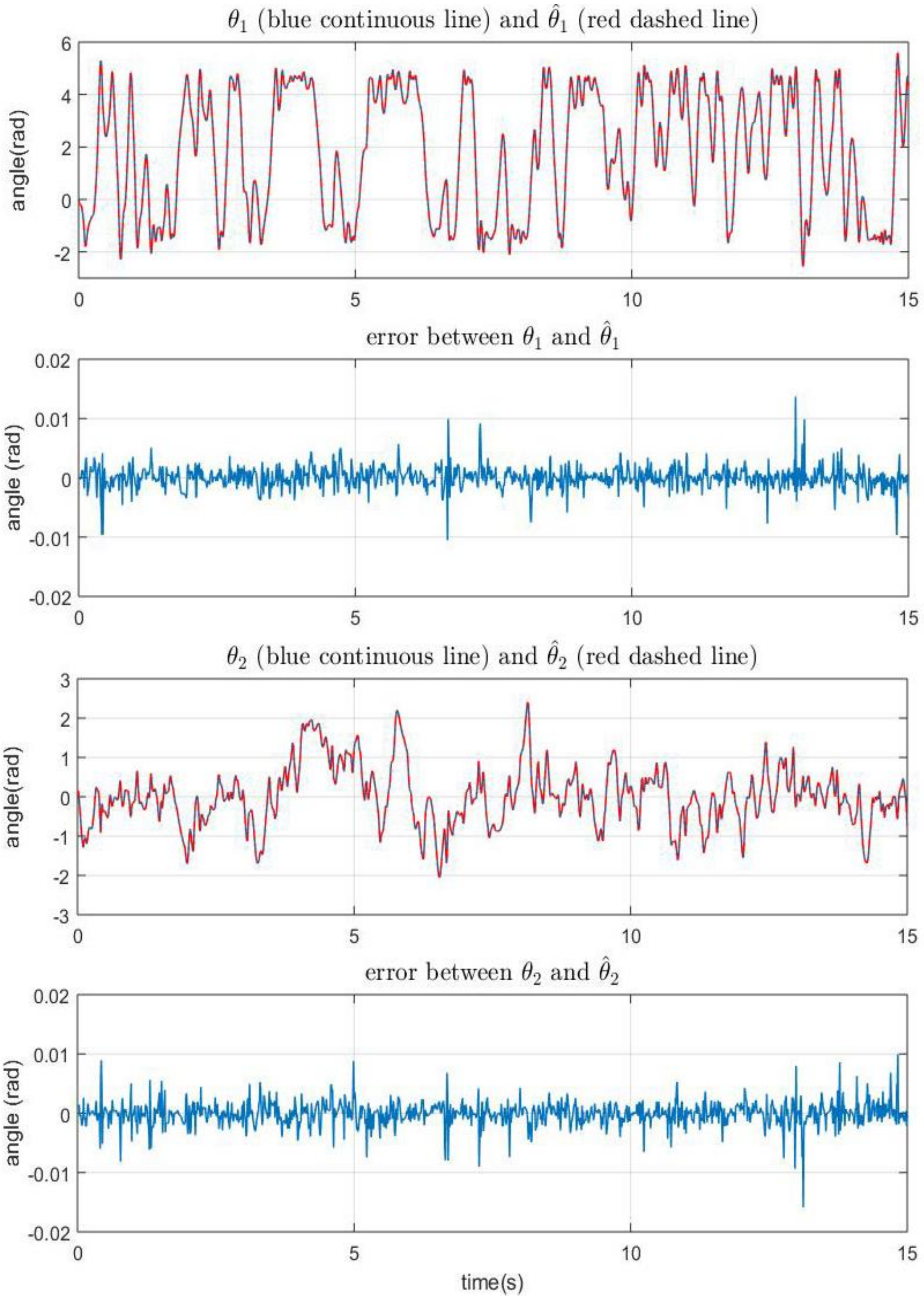


Figure 4. 4 : Test results of the obtained models.

4.3. Controllers implementation

The proposed controllers (NNMPC-TLBO, NNMPC-ITLBO with one teacher, NNMPC-ITLBO with two teachers, and NNMPC-ETLBO) are implemented to control the angular position θ_1 and θ_2 of the considered manipulator.

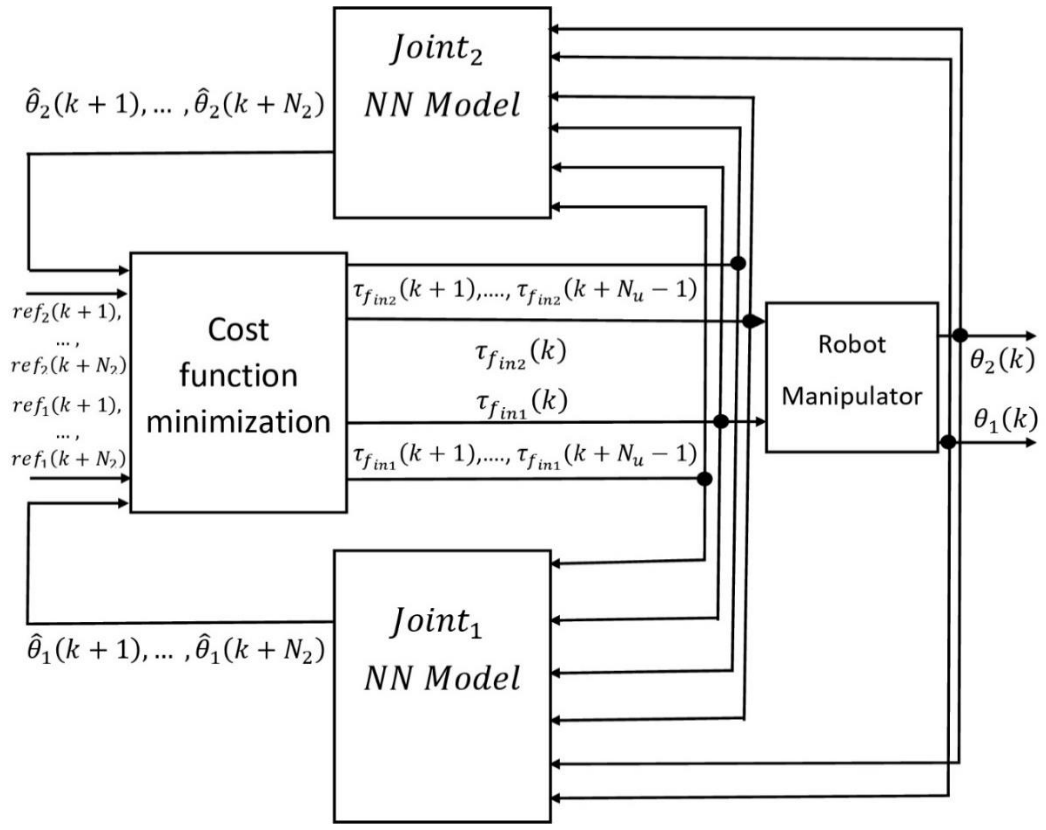
All above-mentioned controllers use the same prediction model, the same control block diagram (figure 4.5) and the same values of the MPC design parameters (table 4.2).

parameter	value	parameter	value
N_u	1	k_{max}	5
N_1	1	n	20
N_2	3	m	2
R	0	Sampling time	0.01s
$\tau_{f_{in1\_min}}$	-50	$\tau_{f_{in2\_min}}$	-50
$\tau_{f_{in1\_max}}$	50	$\tau_{f_{in2\_max}}$	50

Table 4. 2 : MPC design parameters values.

The Mean Cost Value (MCV) is used to compare the performance of the four considered algorithms, it is given by:

$$MCV = \frac{1}{samples} \sum_{k=1}^{samples} J\big(U(k)\big) \tag{4.13}$$



$Joint_2$
$NN\ Model$
$\hat{\theta}_2(k+1),\ldots,\hat{\theta}_2(k+N_2)$
$ref_2(k+1),$
$\ldots,$
$ref_2(k+N_2)$
$ref_1(k+1),$
$\ldots,$
$ref_1(k+N_2)$
Cost function minimization
$\tau_{f_{in2}}(k+1),\ldots,\tau_{f_{in2}}(k+N_u-1)$
$\tau_{f_{in2}}(k)$
$\tau_{f_{in1}}(k)$
$\tau_{f_{in1}}(k+1),\ldots,\tau_{f_{in1}}(k+N_u-1)$
Robot Manipulator
$\theta_2(k)$
$\theta_1(k)$
$Joint_1$
$NN\ Model$
$\hat{\theta}_1(k+1),\ldots,\hat{\theta}_1(k+N_2)$

Figure 4. 5 : Control block diagram of the considered robot arm manipulator.

4.4. Simulation Results

To highlight the control performance of the NNMPC-TLBO, the NNMPC-ITLBO and the NNMPC-ETLBO algorithms, a comparative study of the abovementioned controllers, considering various operating conditions, is carried out. In the first simulation, no output constraints are imposed and two different reference trajectories are used. The obtained results are shown in figure (4.6) for the multistep trajectory and figure (4.7) for the sinusoidal trajectory. Tables (4.3) and (4.4) give the computed average values of the MSE, the MAE and the RMSE over a thousand randomly initialized runs for all considered controllers, for the multistep trajectory and for the sinusoidal trajectory, respectively.

	NNMPC-TLBO	NNMPC-ETLBO	NNMPC-ITLBO($T_n = 1$)	NNMPC-ITLBO($T_n = 2$)
MSE	0.17485	0.16569	0.20023	0.17015
MAE	0.31346	0.28133	0.34842	0.30756
RMSE	0.41816	0.40705	0.44747	0.41249

Table 4. 3 : Average values of MSE, MAE and RMSE in the case of multistep reference trajectory.

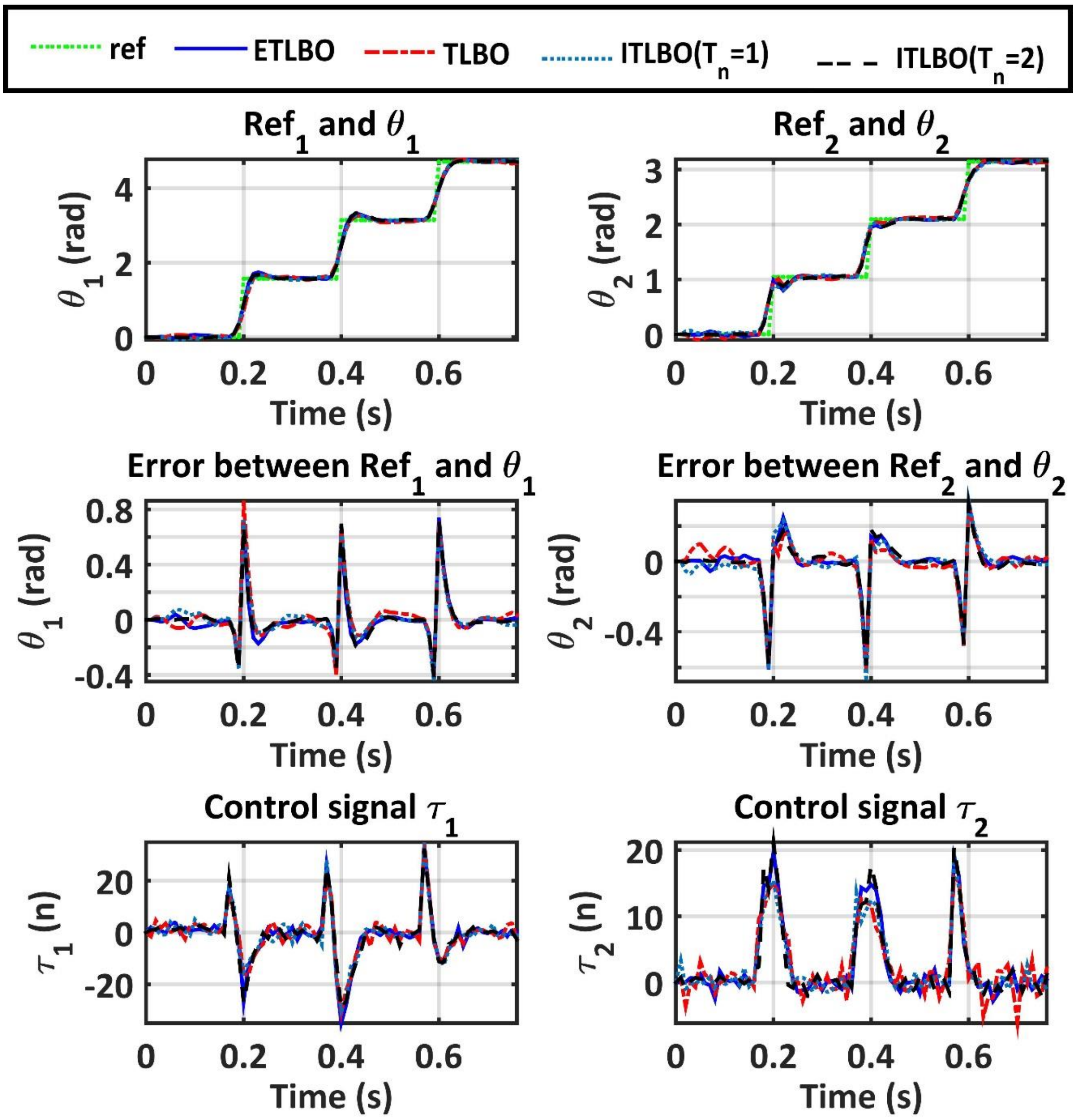


Figure 4. 6 : Control performance using the considered controllers with a multistep trajectory.

	NNMPC-TLBO	NNMPC-ETLBO	NNMPC-ITLBO($T_n = 1$)	NNMPC-ITLBO($T_n = 2$)
MSE	0.02509	0.015511	0.029960	0.020262
MAE	0.19696	0.149291	0.217468	0.179638
RMSE	0.15840	0.124543	0.173090	0.142345

Table 4. 4 : Average values of MSE, MAE and RMSE in the case of sinusoidal reference trajectory.

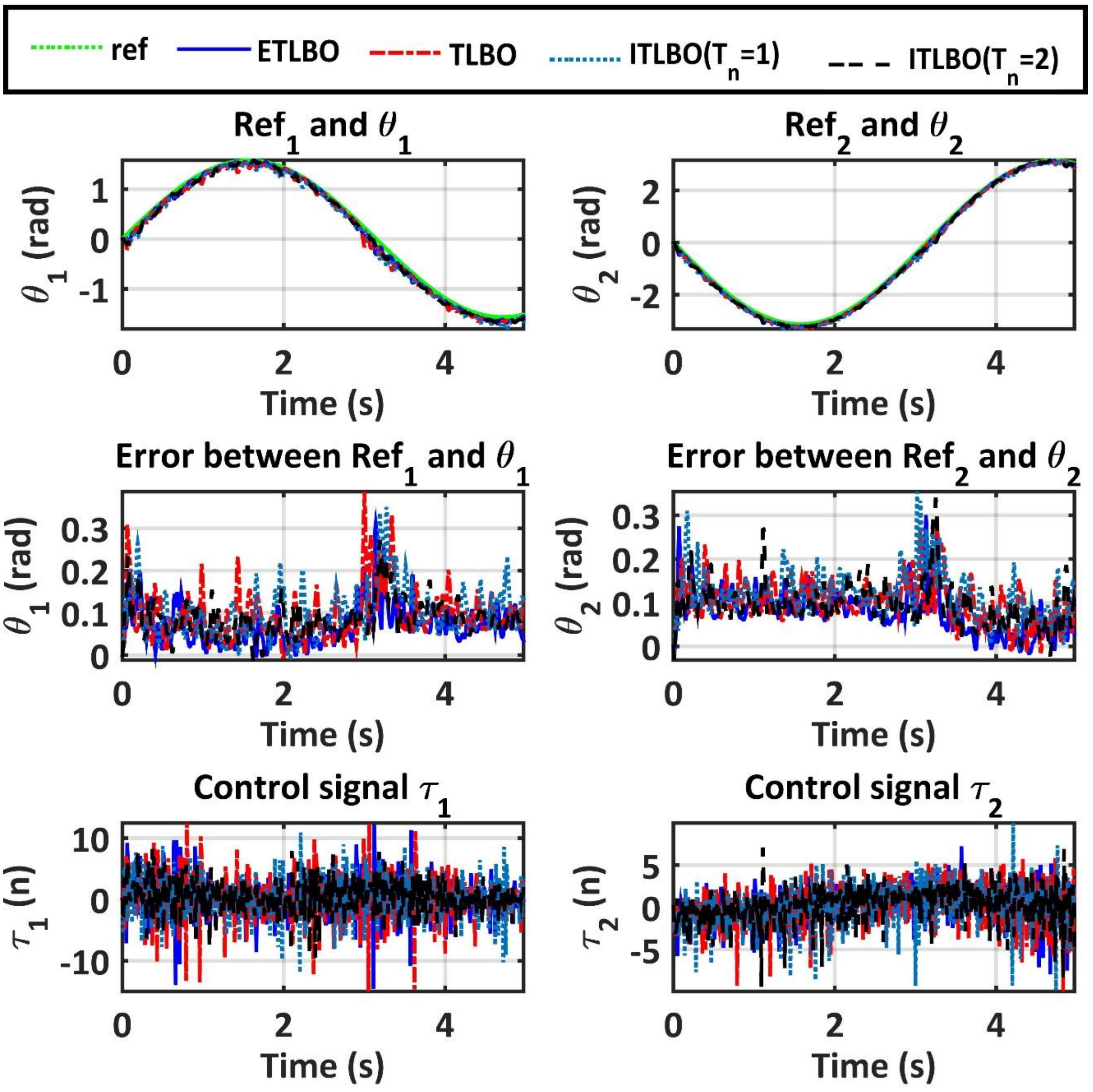


Figure 4. 7 : Control performance using the considered controllers with a sinusoidal trajectory.

From figures (4.6) and (4.7), it can be seen that a good tracking accuracy of the reference trajectories is obtained for all implemented controllers. However, the tracking error for the ETLBO-NNMPC controller is slightly smaller than that of the other considered controllers. From tables (4.3) and (4.4), it is clear that the ETLBO-NNMPC has better tracking accuracy than the other considered controllers.

The aim of the second simulation is to assess the control performance when output constraints limiting the overshoot to no more than 1% are imposed. Step reference trajectories are used and the output-dependent weight function $\Gamma_y(y)$ has the following parameters: $C_{1_y} = 100$, $C_{2_y} = 100$. The design parameters given in table (4.2) are used for all implemented controllers.

Figure (4.8) gives the obtained control performance and table (4.5) gives the MSE average values over 100 randomly initialized runs. From figure (4.8) and table (4.5) it can be seen that all considered controllers handle the imposed constraint well. However, the ETLBO based controller gives better control performance than the other considered controllers.

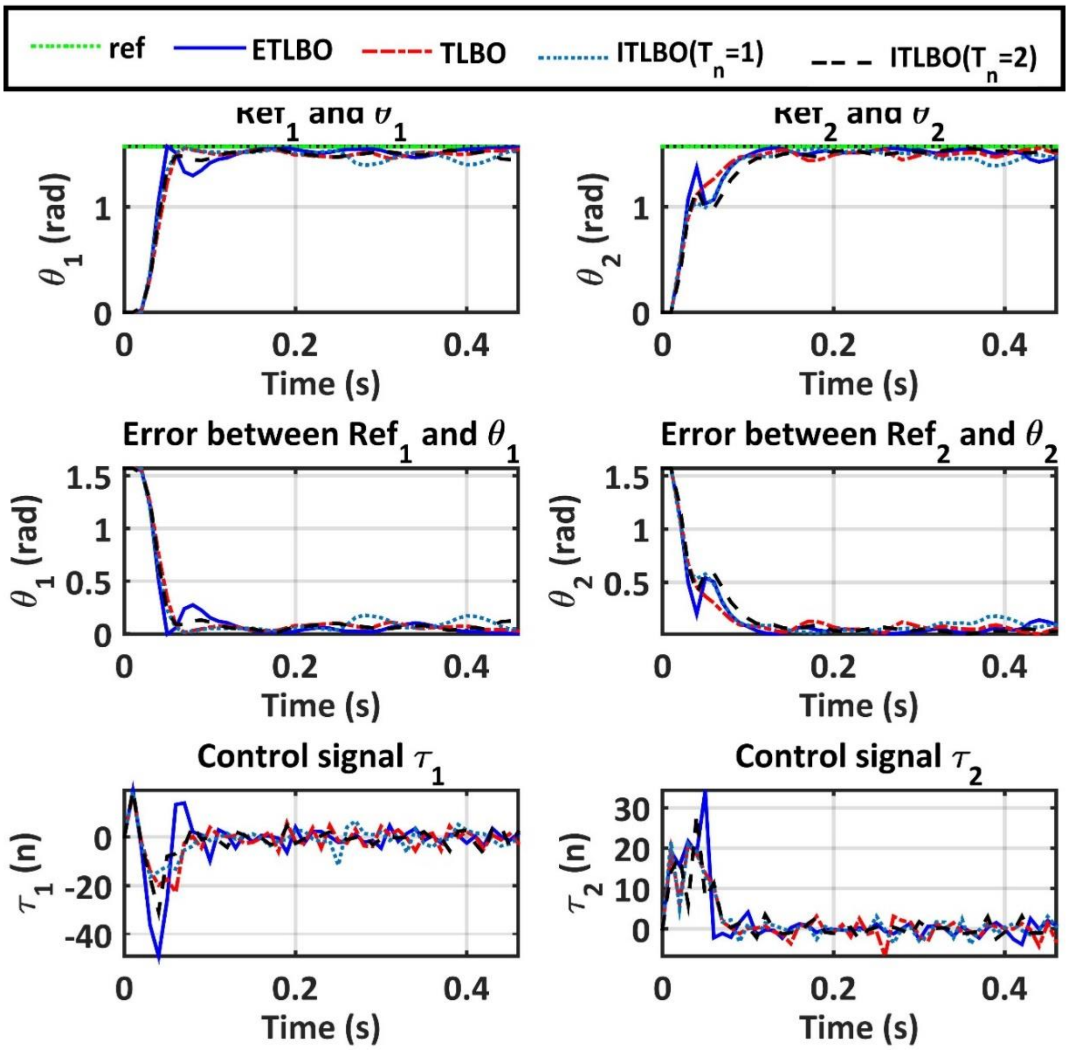


Figure 4. 8 : Control performance using the considered controllers with output constraints.

	NNMPC-TLBO	NNMPC-ETLBO	NNMPC-ITLBO($T_n = 1$)	NNMPC-ITLBO($T_n = 2$)
MSE	0.34440	0.33527	0.34757	0.35682
MAE	0.37612	0.35214	0.40320	0.39112
RMSE	0.58685	0.57903	0.58955	0.59734

Table 4. 5 : Average values of MSE, MAE and RMSE in the case of output constraints.

In the third simulation, to evaluate the control performance of these controllers against different parameters of the optimization algorithm (n, k_{max}), the MCV is evaluated for several values of the population size n and the maximum number of iterations k_{max}. The chosen reference trajectories R_{θ_1} and R_{θ_2} for θ_1 and θ_2 are $R_{\theta_1} = \left[0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\right]$, and $R_{\theta_2} =$

$\left[0, \frac{\pi}{3}, \frac{2\pi}{3}, \pi\right]$, respectively. Each control algorithm is executed 10 times, the average values of MCV are depicted in figure (4.9). It can be seen that the NNMPC-ETLBO gives better control performance than the other considered controllers.

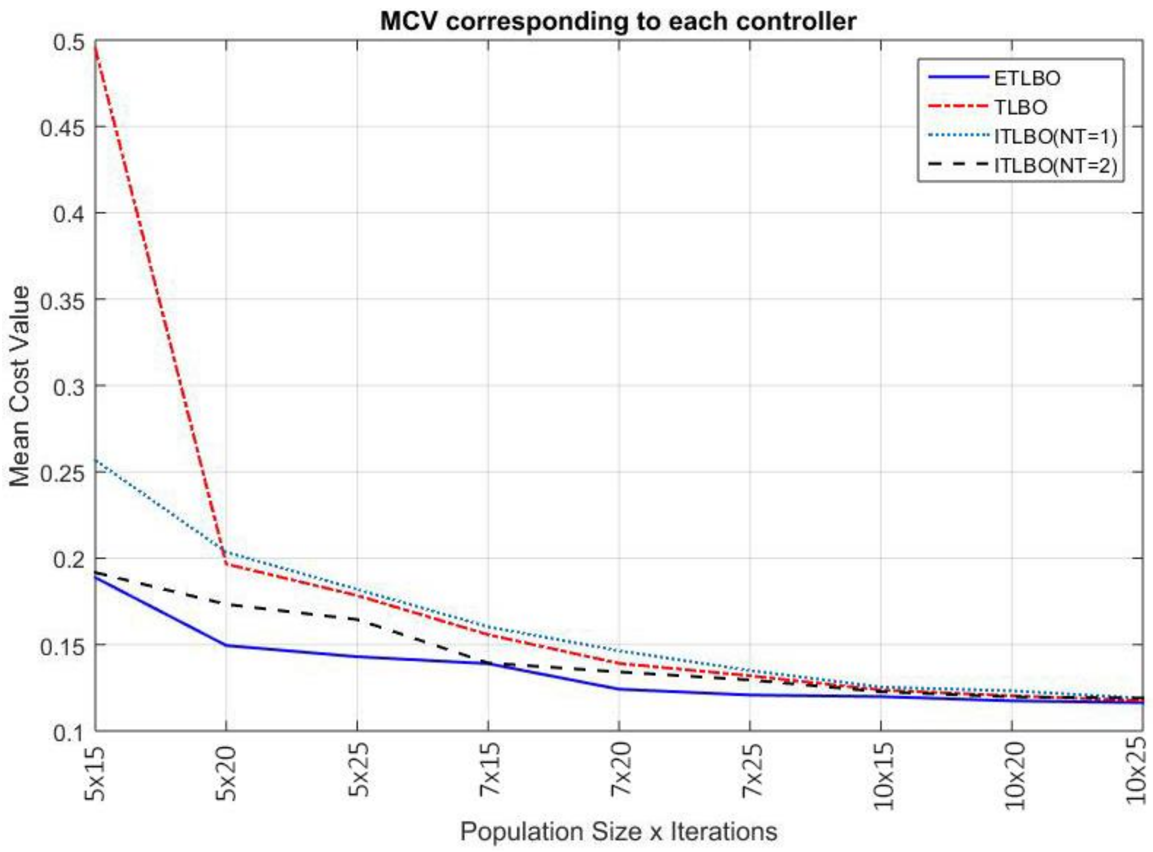


Figure 4. 9 : Average values of MCV corresponding to each controller.

In the final simulation, the maximum number if iterations is fixed ($k_{max} = 50$), and a termination criterion ($\varepsilon = 0.1$) is added to the above mentioned algorithms to stop the optimization process if the minimum cost reach a value less than the termination criterion. The number of iterations, needed to reach the termination criterion, is evaluated for several values of the population size. The same reference trajectories are used. Each control algorithm is executed 10 times, the average values of the iteration number are shown in figure (4.10). It can be seen that the ETLBO-NNMPC algorithm requires few iterations to reach the termination criterion than the other controllers. Therefore, we conclude that the ETLBO-NNMPC algorithm is faster than the other considered controllers.

5. **Experimental study**

To demonstrate further the effectiveness of the proposed controllers the experimental setup shown in figure 4.11 is used to control the speed of an induction motor. In addition to the three-phase squirrel-cage induction motor, this experimental setup contains the following elements : a three-phase generator, a three-phase voltage source inverter, a microcontroller 18f4331 and a single computer board RASPBERRY PI 3B+. The control algorithm is implemented in the single computer board RASPBERRY and the microcontroller is used to generate the six required Pulse Width Modulated (PWM) and measure the motor speed. The control block diagram is given by figure 4.12.

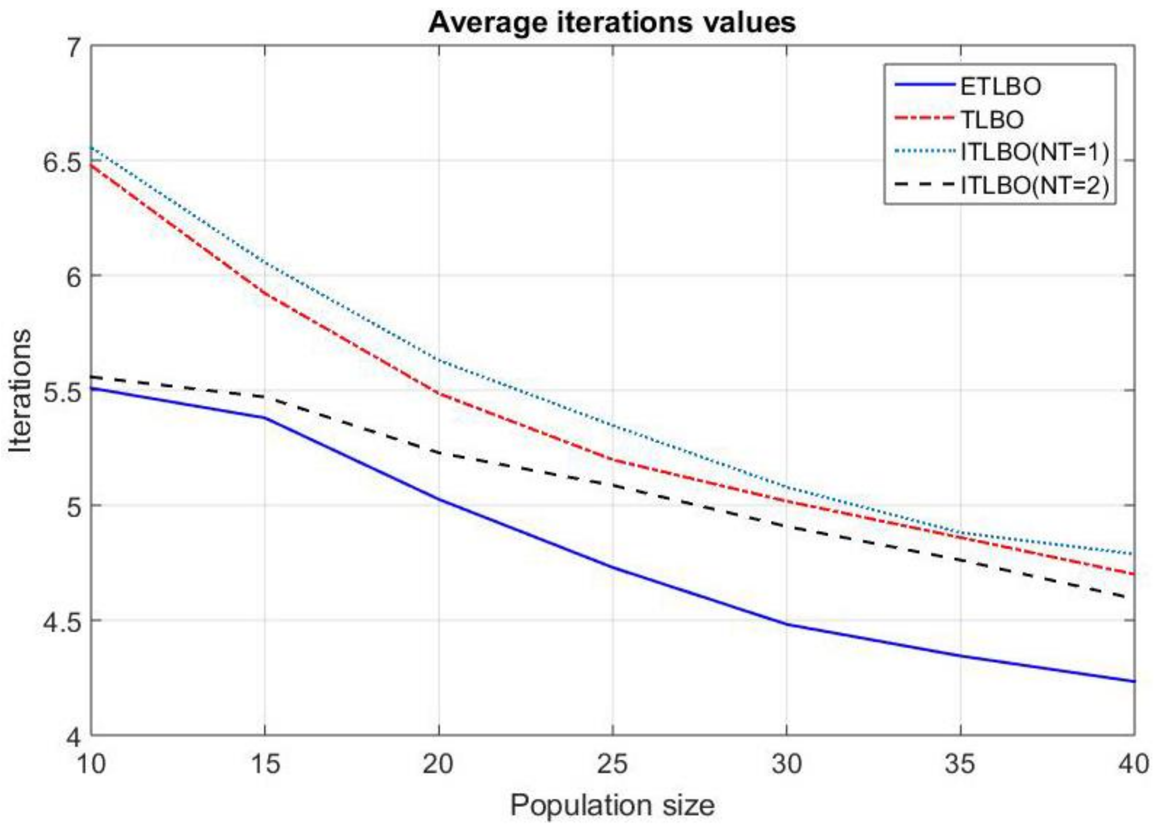


Figure 4. 10 : Iterations number needed to reach the termination criterion for each controller.

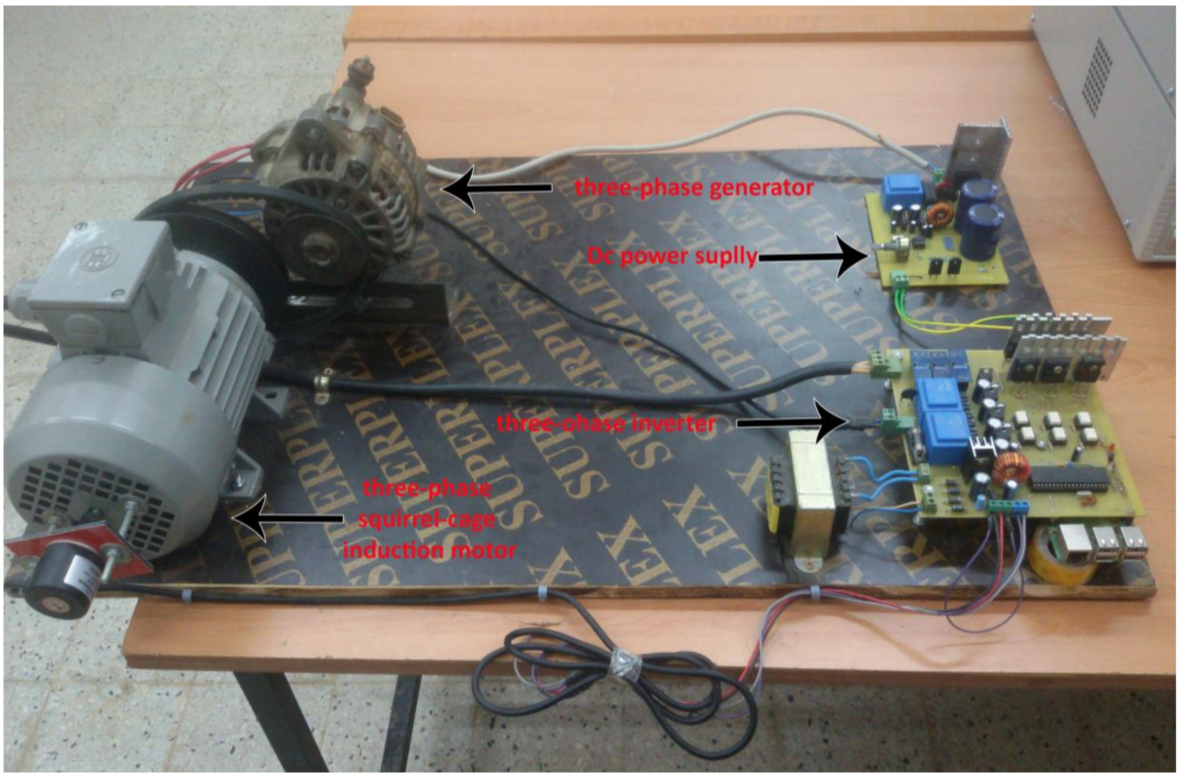
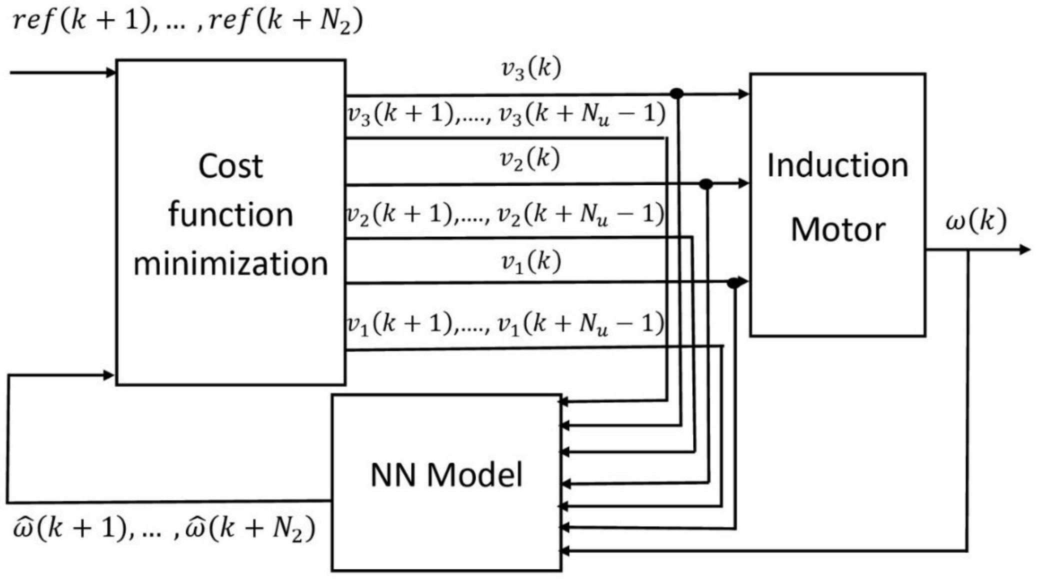


Figure 4. 11 : Experimental set up.



$ref(k+1), \ldots, ref(k+N_2)$
Cost function minimization
$v_3(k)$
$v_3(k+1), \ldots, v_3(k+N_u-1)$
$v_2(k)$
$v_2(k+1), \ldots, v_2(k+N_u-1)$
$v_1(k)$
$v_1(k+1), \ldots, v_1(k+N_u-1)$
Induction Motor
$\omega(k)$
NN Model
$\widehat{\omega}(k+1), \ldots, \widehat{\omega}(k+N_2)$

Figure 4. 12 : The induction motor control block diagram.

5.1. Modeling of the induction motor

A neural network model, for the considered induction motor is derived using the experimental collected data. This model is a simple static MLP with an input layer containing four inputs $[v_1(k), v_2(k), v_3(k), \omega(k-1)]$, a hidden layer of four neurons with sigmoid activation function and an output layer that contains one linear neuron representing the estimated angular speed $\widehat{\omega}(k)$. $v_1(k), v_2(k)$ and $v_3(k)$ are the applied voltages on the motor at the sampling time k and $\omega(k-1)$ is the measured angular speed of the motor. The response of the obtained model to the input test is given by figure 4.13, where it can be seen the modeling error is quite small. The RMSE and the R^2 values of the obtained model are: 11.19669 and 0.99976, respectively. Since the amplitude of the system output ranges between -1500tr/min and 1500tr/min, the RMSE value is acceptable. The R^2 value is close to one, which indicates that the model has good accuracy.

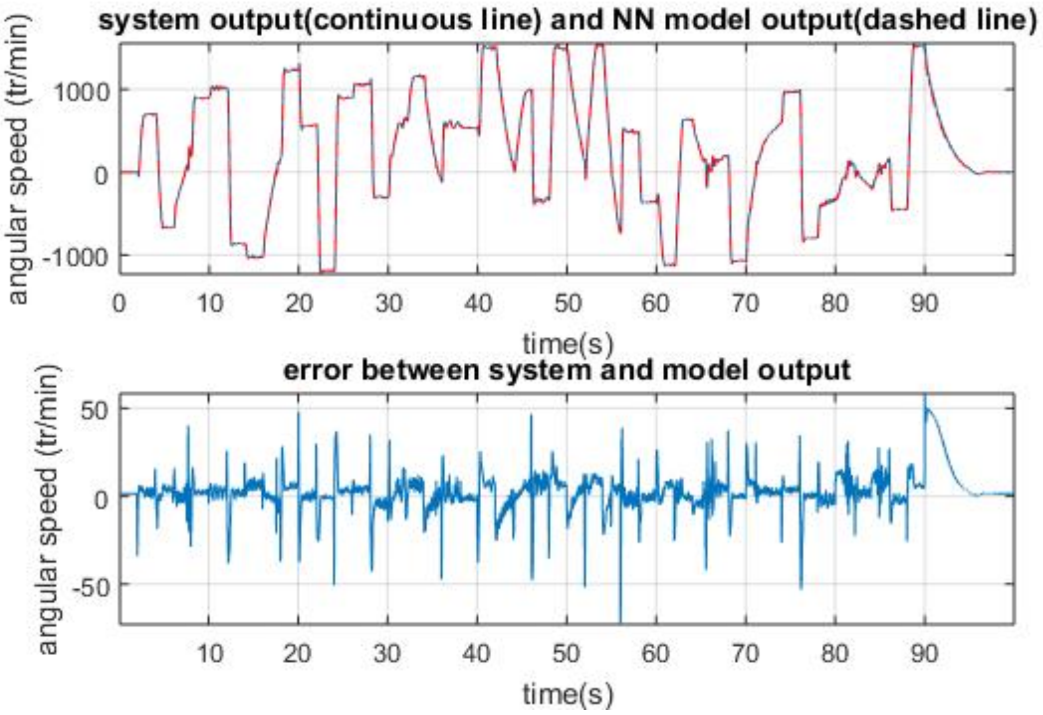


Figure 4. 13 : Test results of the induction motor model.

5.2. Controllers implementation

The NNMPC-TLBO, the NNMPC-ITLBO ($T_n = 2$) and the NNMPC-ETLBO algorithms are implemented using the values of the design parameters given in table (4.6) and the obtained neural network model. The control objective is to force the motor speed to track two different reference trajectories; the sinusoidal and the multistep trajectories. The results are given in figure (4.14) and (4.15). Table (4.7) and (4.8) give the values of the MSE, MAE and RMSE using all considered controller in both cases (multistep and sinusoidal trajectories).

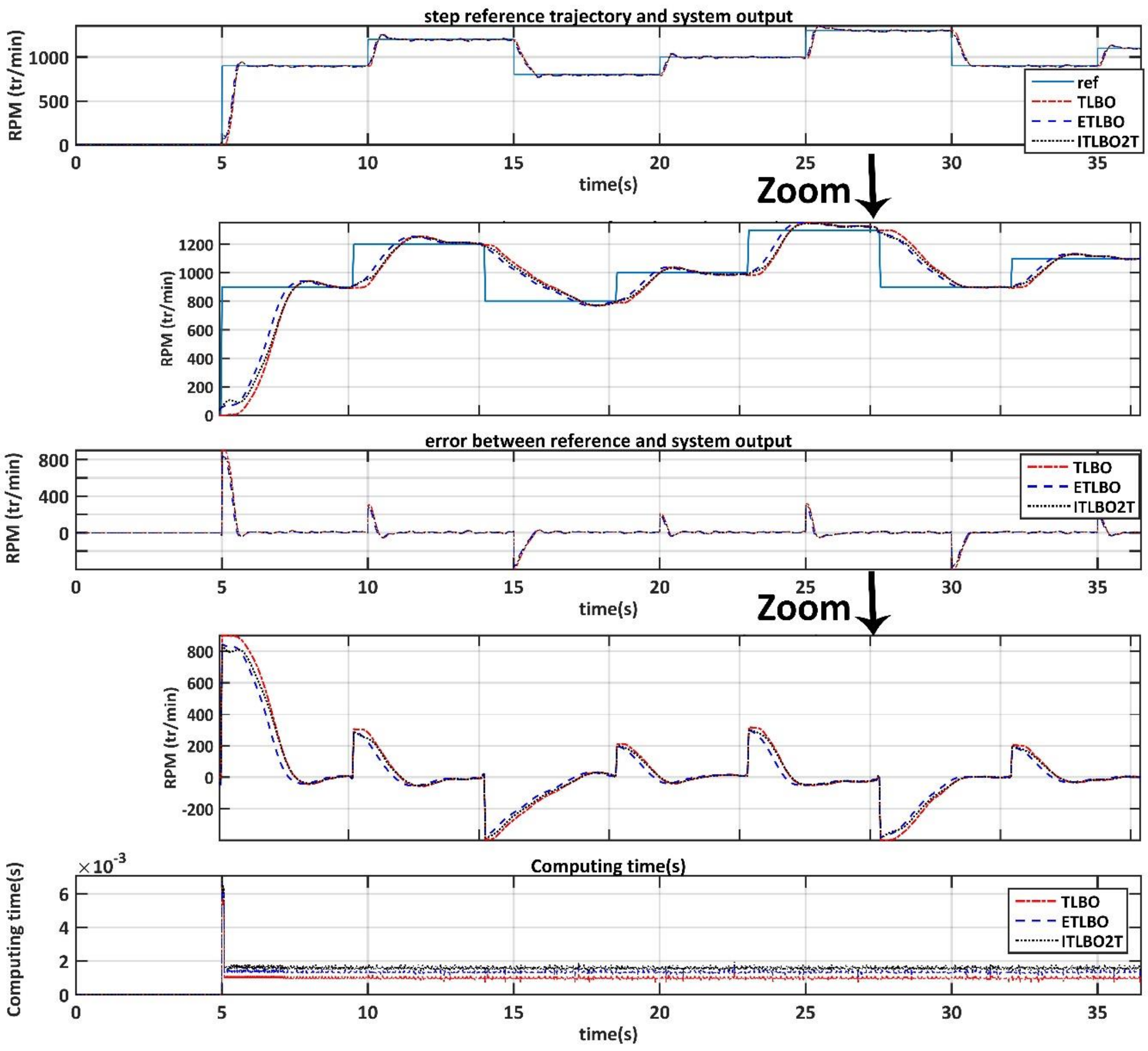


Figure 4. 14 : Control performance in case of the multistep trajectory for the induction motor.

parameter	value	parameter	value
N_u	1	k_max	10
N_1	1	n	10
N_2	3	m	3
R	10	Sampling time	0.01s

Table 4. 6 : Values of the design parameter.

From figures (4.14) and (4.15), it can be seen that a good tracking accuracy of the reference trajectories is obtained for all implemented controllers. However, the tracking error for the ETLBO-NNMPC controller is slightly smaller than that of the other considered controllers. From tables (4.7) and (4.8), it is clear that the ETLBO-NNMPC has better tracking accuracy than the other considered controllers.

In the case of all considered controllers, the maximal value for the overshoot does not exceed 4% for both (step and sinusoidal) reference trajectories, the maximum values of the computing time are : 7.225ms and 6.402ms successively, which indicates that the proposed controllers can be implemented in real time to control systems with fast dynamics.

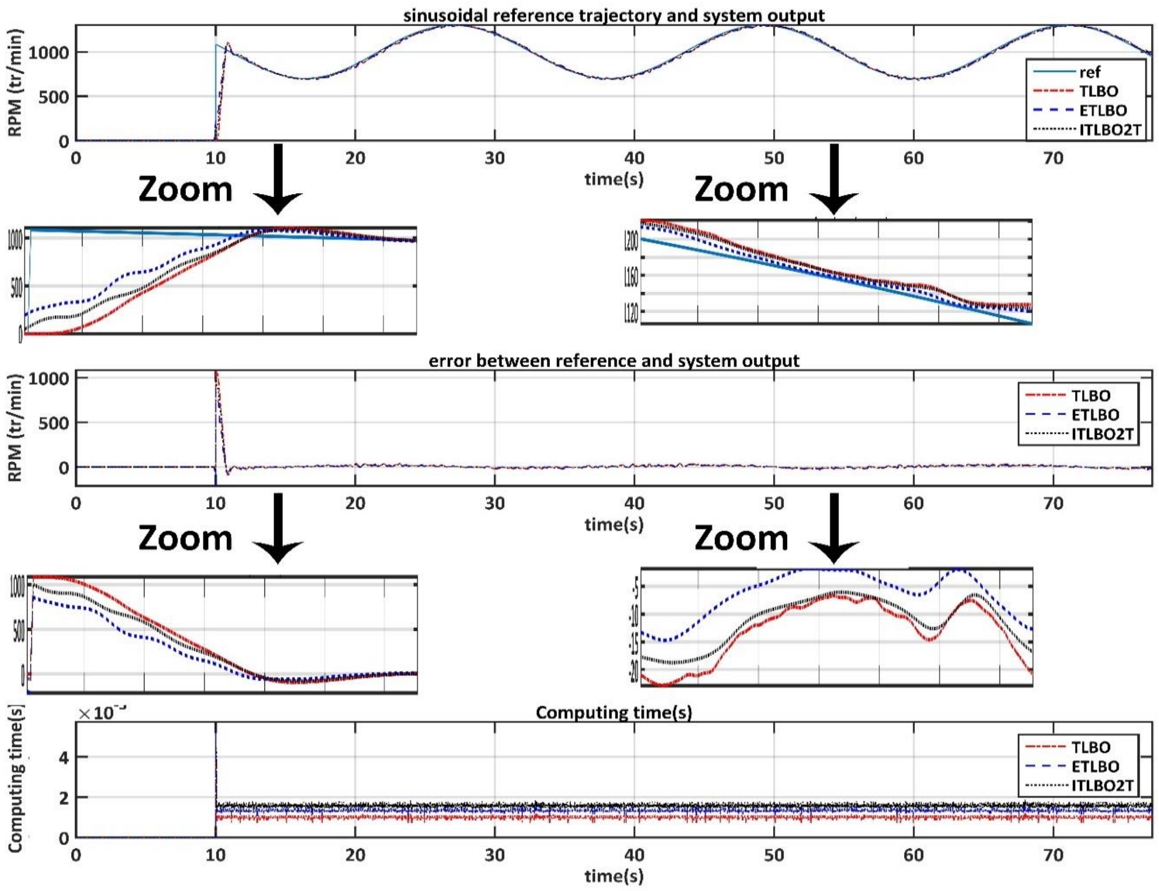


Figure 4. 15 : Control performance in case of the sinusoidal trajectory for the induction motor.

	NNMPC-TLBO	NNMPC-ETLBO	NNMPC-ITLBO($T_n = 2$)
MSE	10414	7697.4	8625.1
MAE	29.6762	24.8852	26.9914
RMSE	102.0485	87.7348	92.8716

Table 4. 7 : Values of MSE, MAE and RMSE in the case of a multistep trajectory for the induction motor control.

	NNMPC-TLBO	NNMPC-ETLBO	NNMPC-ITLBO($T_n = 2$)
MSE	11038	5445.2	8173.4
MAE	38.5634	24.1720	33.1700
RMSE	105.0597	73.7919	90.4066

Table 4. 8 : Values of MSE, MAE and RMSE in the case of a sinusoidal trajectory for the induction motor control.

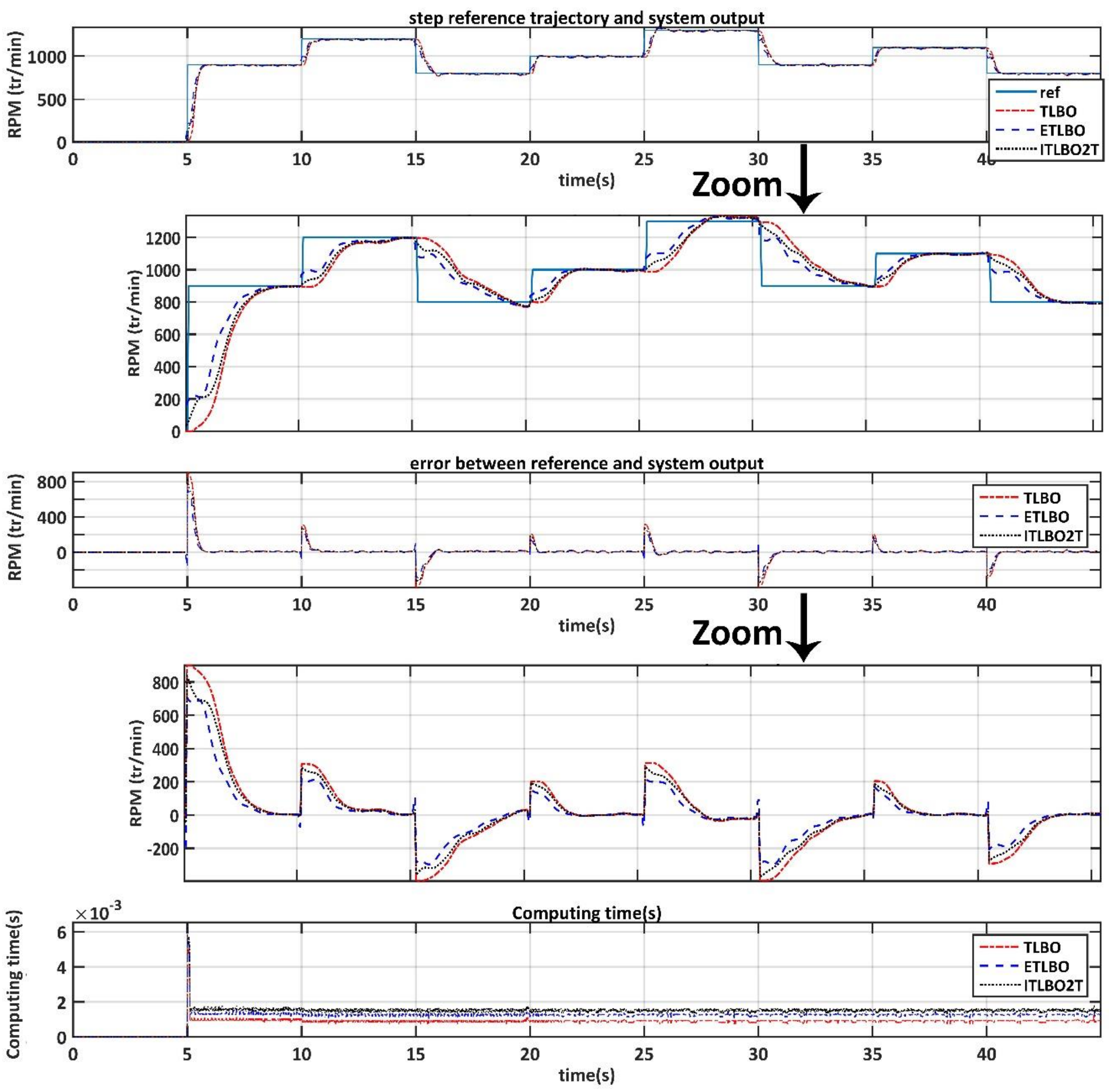


Figure 4. 16 : Control performance of the induction motor in the presence of output constraints.

In a second experiment, to further reduce the observed overshoot, a constraint on the output, limiting the overshoot to no more than 2%, is included. A multistep reference trajectory and the parameter $C_y = 100$ of the output-dependent weight function $\Gamma_y(y)$ are used in this experiment. The obtained control results are given by figure (4.16), where it can be seen that the overshoot value does not exceed 2%, and the maximum value of the computing time is : 5.839ms for all considered controllers.

	NNMPC-TLBO	NNMPC-ETLBO	NNMPC-ITLBO($T_n = 2$)
MSE	10684	5304.3	7721.8
MAE	35.6002	24.8474	30.3787
RMSE	103.3654	72.8303	87.8739

Table 4. 9 : Values of MSE, MAE and RMSE in the presence of output constraints for the induction motor control.

Table (4.9) gives the values of the MSE, the MAE and the RMSE for the released experiments. From this table, we can conclude that all proposed controllers give good control performances and the imposed constraints are respected. However, the NNMPC-ETLBO algorithm gives the best control performance.

6. Conclusion

Within this chapter, the formulation of the constrained neural network predictive control based on the meta-heuristic algorithms has been given along with three proposed control algorithms. It has been shown, through several simulation studies, that the proposed controllers can be successfully used to control highly constrained nonlinear systems. Indeed, the control of a coupled multivariable mechanical system was considered. The obtained results have shown that the proposed controllers give good performance in terms of the tracking accuracy, the overshoot amplitude and the settling time. In addition, it has been shown, using experimental studies, that the proposed controllers give good control performance. The proposed algorithms can be successfully used to control different classes of nonlinear systems. Indeed, the control of a coupled multivariable mechanical system, and an electrical machine, were considered.

CONCLUSION

Using artificial intelligence tools, such as neural networks and meta-heuristic optimization methods, the aim of this thesis was to develop efficient, simple, and robust control algorithms that can give satisfactory performance with a large class of nonlinear systems. In fact, neural networks have been used extensively in the field of identification and control of nonlinear systems. The extensive research carried out in this field has proven the feasibility and the efficiency of control systems based on neural networks. The secret to the success of neural networks lies in the fact that any nonlinear system can be modeled, with a given precision, using a simple neural network with learning and generalization capabilities. On the other hand, the emergence of several meta heuristic optimization methods has allowed to consider more complex optimization problems for which numerical methods cannot give acceptable solutions. The numerical approach can quickly reach its limits when the system to be controlled or the constraints to be respected become complex, or when the optimization of the system operating is required. Recently, the meta heuristic approach was successfully used in many control applications.

In this thesis work, we started by studying several meta heuristic optimization methods, such as genetic algorithms, particle swarm optimization and several versions of the teaching learning based optimization method. This study allowed us to examine the limits and advantages of each algorithm and to propose an improvement for the teaching learning based optimization method. Dues to its attractive proprieties the TLBO, a meta heuristic method, has been used in many engineering applications and given satisfactory results. In fact, except the common control parameters (population size and number of generations) the TLBO algorithm, unlike to other meta heuristic algorithms, does not require any algorithm specific-parameters. Obviously, improper tuning of algorithm-specific parameters either increases the computational effort or yields a local optimal solution. The improvement made to the TLBO algorithm consisted in replacing the random selecting process of the students' pairs in the learners' phase by a new strategy based on the grade of each student obtained during the optimization process. This modification has allowed improving the convergence rate and the exploitation quality of the algorithm without altering its complexity. The convergence rate and the efficiency of the modified algorithm (ETLBO) have been assessed by considering eight well-known benchmark functions. The obtained results have showed that the proposed ETLBO algorithm outperforms the other considered algorithms; namely the original TLBO, the ITLBO with one and two teachers, and the w-PSO algorithm.

Furthermore, the study carried out on neural networks and their application to the identification and the control of nonlinear systems has allowed discovering a type of these networks with a simple architecture and a simple learning algorithm. This neural network, called Fourier series neural network, is not widely used in the field of the identification and control of nonlinear systems. The main raison for which the use of this kind of neural networks could be very useful is the simplicity of their training algorithm. Indeed, Fourier series neural networks can be trained using the simple Delta rule algorithm. Therefore, it can be easily incorporated in adaptive control systems.

The research work done throughout this thesis has allowed developing several control algorithms; namely the adaptive neural network PID controller, the adaptive Fourier series neural network PID controller, the neural network predictive control using the TLBO algorithm, the neural network predictive control using the Improved TLBO and the neural network predictive control using the ETLBO.

In addition to the proposed adaptive neural network PID and the adaptive Fourier series neural network PID controllers, the PSO based adaptive PID controller, where the PSO algorithm was used to online optimize the PID controller gains, has been considered. In the adaptive neural network PID, the PID controller parameters are obtained from a multilayer perceptron (MLP) neural network and their values are online tuned using the back propagation method. The adaptive Fourier series neural network PID controller uses two FSNN; the first one allows estimating the system Jacobian and the second is used to obtain and online adjust the PID controller gains. The stability of the adaptive Fourier series neural network PID controller has been proved using the small gain theorem. To assess the effectiveness of the ANNPID, the AFSNNPID and the PSO-based PID controllers in controlling highly nonlinear systems, the control of the continuous stirred tank reactor and the 3-DOF robot arm manipulator, through simulation and experimental studies, has been investigated. The simulation and the experimental results have shown that these controllers give good control performance in terms of the tracking accuracy and the robustness against external disturbances and dynamic system variation. However, the proposed AFSNNPID controller do not require a large computing time, which allows it to be used in several real time applications. Indeed, the AFSNNPID controller has a simple design procedure and can be used to control any nonlinear system.

Model predictive control is a sophisticated control technique that is widely used in industrial applications and still continues to raise interest of several researchers. In this thesis, the formulation of the constrained neural network predictive control based on the meta-heuristic algorithms has been given along with three proposed control algorithms; namely the TLBO based NNMPC, the ITLBO based and the ETLBO based NNMPC. It has been shown, through several simulation studies and experimental study, that the proposed controllers can be successfully used to control in real time highly nonlinear systems. Indeed, the control of a coupled multivariable mechanical system (the robot arm manipulator) and an electrical machine (the induction motor) were considered. The obtained results have shown that the proposed controllers give good performance in terms of the tracking accuracy, the overshoot amplitude and the settling time.

Throughout this thesis, several artificial intelligent based control algorithms have been discussed and analyzed. The future possible research works are:

- Instead of analyzing the stability of the proposed adaptive Fourier series neural network PID controller using the small gain theorem, the Lyapunov approach should be used. The small gain theorem proves the BIBO stability and assumes that the system and the controller are stable in an open loop architecture. However, Lyapunov approach proves the asymptotic stability without any assumptions.
- Improving the proposed version of the TLBO algorithm by using the concept of multi-teachers instead of using a single one in the teacher phase.
- To improve the tracking accuracy of the MPC, the Fourier series neural network could be used as a prediction model.

Incorporating the stability analysis in the formulation of the proposed nonlinear model predictive control using the enhanced teaching learning based optimization algorithm.

LIST OF ABBREVIATIONS

ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
AFSNNPID	Adaptive Fourier Series Neural Network PID
AI	Artificial Intelligence
ANFIS	Adaptive Network based Fuzzy Inference System
ANNPID	Adaptive Neural Network based PID
CMAC	Cerebellar Model Arithmetic Computer
CSTR	Continuous Stirred Tank Reactor
DMC	Dynamic Matrix Control
EA	Evolutionary Algorithm
ETLBO	Enhanced Teaching Learning Based Optimization
FL	Fuzzy Logic
FLC	Fuzzy Logic Control
FMPC	Fuzzy model Based NMPC
FNNC	Fuzzy Neural Network Control
FNNMPC	Fuzzy Neural Network based NMPC
FPID	Fuzzy PID
FSF	Full State Feedback
FuNe	Fuzzy Neural system
GA	Genetic Algorithms
GARIC	Generalized Approximate Reasoning-based Intelligent Control
GRASP	Greedy Randomized Adaptive Search Procedure
GWO	Gray Wolf Optimizer
INNFLC	Integrated Neural Network based Fuzzy Logic Control
ITLBO	Improved Teaching Learning Based Optimization
LMS	Least Mean Square
LQG	Linear Quadratic Gaussian
LQR	Linear Quadratic Regulator
MAC	Model Algorithmic Control
MAE	Mean Absolute Error
MCV	Mean Cost Value
MIMO	Multiple Inputs Multiple Outputs
MISO	Multi Inputs Single Output
MLP	Multi Layer Perceptron
MPC	Model Predictive Control
MRAC	Model Reference Adaptive Control
MSE	Mean Squared Error
NDTS	Nonlinear Discrete-Time Systems
NEFCON	NEuro Fuzzy CONtrol
NGPC	Nonlinear Generalized Predictive Control

NMPC	Nonlinear Model Predictive Control
NN	Neural Networks
NNC	Neural Network Control
NNMPC	Neural Network Based NMPC
NNPID	Neural Network based PID
PD	Proportional Derivative
PID	Proportional Integrate Derivative
PSO	Particle Swarm Optimization
PWM	Pulse Width Modulated
RMSE	Root Mean Square of the modeling Error
TLBO	Teaching Learning Based Optimization

REFERENCES

[1] Maxwell, J.C., "I. On governors", Proceedings of the Royal Society of london, London, (1868), 270–83.

[2] Routh, E.J., "On Laplace’s three particles, with a supplement on the stability of steady motion", Proceedings of the London Mathematical Society, London, (1874), 86–97.

[3] Routh, E.J., "Stability of a dynamical system with two independent motions", Proceedings of the London Mathematical Society, London, (1873), 97–9.

[4] Routh, E.J., "A Treatise on the Stability of Motion", Macmillima, London (1877).

[5] Hurwitz, A., "Ueber die Bedingungen, unter welchen eine Gleichung nur Wurzeln mit negativen reellen Theilen besitzt", Mathematische Annalen, V.46, n°2, (1895), 273–284.

[6] Hurwitz, A., "On the conditions under which an equation has only roots with negative real parts". Selected papers on mathematical trends in Control Theory, V. 65, (1964), 273–284.

[7] Minorsky, N., "Directional stability of automatically steered bodies", Journal of the American Society for Naval Engineers, V. 34, n° 2, (1922), 280–309.

[8] Sontag, E.D., "Mathematical control theory: deterministic finite dimensional systems", Springer Science & Business Media, (2013).

[9] Kwakernaak, H. and Sivan, R., "Linear optimal control systems", Wiley-interscience, New York, (1972).

[10] Astrom, K.J., "Introduction to Stochastic Control Theory", Courier Corporation, (1970).

[11] Åström, K.J., "Introduction to stochastic control theory". Courier Corporation, (2012).

[12] Richalet, J. Rault, A. Testud, J.L.and Papon, J., "Model predictive heuristic control", Applications to industrial processes, Automatica, V. 14, n° 5, (1978), 413–428.

[13] Cutler, C.R. and Ramaker, B.L., "Dynamic Matrix Control - a Computer Control Algorithm". Joint Automatic Control Conference, V. 17, (1980), 72-82.

[14] Hamid, N.H.A. Kamal, M.M. and Yahaya, F.H., "Application of PID controller in controlling refrigerator temperature", International Colloquium on Signal Processing & Its Applications. IEEE, (2009), 378–84.

[15] Aguilar, R. Poznyak, A. Martínez-Guerra, R. and Maya-Yescas, R., "Temperature control in catalytic cracking reactors via a robust PID controller", Journal of Process Control, V. 12, n° 6, (2002), 695–705.

[16] Wang, Y. Geng, Y. Yan, Y. Wang, J. and Fang, Z., "Robust model predictive control of a micro machine tool for tracking a periodic force signal", Optimal Control Applications and Methods, V. 41, n° 6, (2020), 2037–2047.

[17] Pinheiro, T.C.F. and Silveira, A.S., "Constrained discrete model predictive control of an arm-manipulator using Laguerre function". Optimal Control Applications and Methods, V. 42, n° 1, (2021), 160–179.

[18] Rehman, O.U. Fidan, B. and Petersen, I.R., "Uncertainty modeling and robust minimax LQR control of multivariable nonlinear systems with application to hypersonic flight", Asian Journal of Control, V. 14, n° 5, (2012), 1180–1193.

[19] Lee, J. Kim, J.S. and Shim, H., "Disc margins of the discrete-time LQR and its application to consensus problem", International Journal of Systems Science, V. 43, n° 10, (2012); 1891–1900.

[20] Grimble, M.J., "Two and a half degrees of freedom LQG controller and application to wind turbines", IEEE transactions on automatic control, V. 39, n° 1, (1994), 122–127.

[21] Gawronski, W.K. Racho, C.S. and Mellstrom, J.A., "Application of the LQG and feedforward controllers to the deep space network antennas". IEEE Transactions on Control Systems Technology, V. 3, n° 4, (1995), 417–421.

[22] Jingqing, H., "Nonlinear PID controller", Acta Automatica Sinica, V. 20, n° 4, (1994), 487–490.

[23] Seraji, H., "A new class of nonlinear PID controllers with robotic applications", Journal of Robotic Systems, V. 15, n° 3, (1998), 161–181.

[24] Prakash, J. and Srinivasan, K., "Design of nonlinear PID controller and nonlinear model predictive controller for a continuous stirred tank reactor", ISA transactions, V. 48, n° 3, (2009), 273–282.

[25] Atherton, D.P. Benouartes, M. and Nanka-Bruce, O., "Design of nonlinear PID controllers for nonlinear plants", IFAC Proceedings Volumes, V. 26, n° 2, (1993), 125–128.

[26] Drakunov, S. V. and Utkin, V.I., "Sliding mode control in dynamic systems", International Journal of Control, V. 55, n° 4, (1992), 1029–1037.

[27] Shtessel, Y. Edwards, C. Fridman, L. and Levant, A., "Sliding mode control and observation", Springer, New York, (2014).

[28] Young, K.D. Utkin, V.I. and Ozguner, U., "A control engineer’s guide to sliding mode control", IEEE transactions on control systems technology, V. 7, n° 3, (1999), 328–342.

[29] Bellman, R. Glicksberg, I. and Gross, O., "On the “bang-bang” control problem", Quarterly of Applied Mathematics, V. 14, n° 1, (1956), 11–18.

[30] Wonham, W.M. and Johnson, C.D., "Optimal bang-bang control with quadratic performance", index, (1964), 107–115.

[31] Käpernick, B. and Graichen, K., "Nonlinear model predictive control based on constraint transformation", Optimal Control Applications and Methods, V. 37, n° 4, (2016), 807–828.

[32] Conceicao, A.S. Moreira, A.P. and Costa, P.J., "A nonlinear model predictive control strategy for trajectory tracking of a four-wheeled omnidirectional mobile robot", Optimal Control Applications and Methods, V. 29, n° 5, (2008), 335–352.

[33] Chen, W.H. Balance, D.J. and Gawthrop, P.J., "Nonlinear generalised predictive control and optimal dynamical inversion control", IFAC Proceedings Volumes, V. 32, n° 2, (1999), 2540–2545.

[34] Feng, X. Yu, T. and Wang, J., "Nonlinear GPC with In-place Trained RLS-SVM Model for DOC Control in a Fed-batch Bioreactor", Chinese Journal of Chemical Engineering, V. 20, n°5, (2012), 988–994.

[35] Miyamoto, H. Kawato, M. Setoyama, T. and Suzuki, R. "Feedback-error-learning neural network for trajectory control of a robotic manipulator", Neural Networks, V. 1, n° 3, (1988), 251–265.

[36] Mumme, D.C. and Chick, D.R., "Design of a neural-network control system", EG and G Idaho, Idaho Falls, USA, (1988).

[37] Li, W. and Slotine, J.E., "Neural Network Control of Unknown Nonlinear Systems", American Control Conference, (1989), 1136–1141.

[38] Sanner, R.M. and Slotine, J.E., "Gaussian networks for direct adaptive control", American control conference, (1991), 2153–2159.

[39] Chen, F.C. and Khalil, H.K., "Adaptive control of nonlinear systems using neural networks", International journal of control, V. 55, n° 6, (1992), 1299–1317.

[40] Chen, F.C., "Back-propagation neural networks for nonlinear self-tuning adaptive control", IEEE control systems Magazine, V. 10, n° 3, (1990), 44–48.

[41] Ryu, Y.S. and Oh, S.Y., "A neural network architecture for dynamic control of robot manipulators", Control Robot System Society: Conference Papers, (1989), 1113–1119.

[42] Antsaklis, P.J., "Neural networks for control systems", IEEE Transactions on Neural Networks, V. 1, n° 2, (1990), 242–244.

[43] Miller, W.T. Glanz, F.H. and Kraft, L.G., "Cmas: An associative neural network alternative to backpropagation", Proceedings of the IEEE, V. 78, n° 10, (1990), 1561–1567.

[44] Kraft, L.G. and Campagna, D.P., "A comparison between CMAC neural network control and two traditional adaptive control systems", IEEE Control Systems Magazine, V. 10, n° 3, (1990), 36–43.

[45] Wang, G.J. and Miu, D.K., "Unsupervising adaption neural-network control", International Joint Conference on Neural Networks, (1990), 421–428.

[46] Johnson, M.A. and Leahy, M.B., "Adaptive model-based neural network control", IEEE International Conference on Robotics and Automation, (1990), 1704–1709.

[47] Mb, L. Johnson, M.A. Bossert, D.E. and Lamont, G.B., "Robust model-based neural network control", IEEE International Conference on Systems Engineering, (1990), 343–346.

[48] Yamada, T. and Yabuta, T., "Nonlinear neural network controller for dynamic system", Annual Conference of IEEE Industrial Electronics Society, (1990), 1244–1249.

[49] Okuma, S. Ishiguro, A. Furuhashi, T. and Uchikawa, Y., "A neural network

compensator for uncertainties of robotic manipulators", IEEE Transactions on Industrial Electronics, V. 39, n° 6, (1990), 3303–3307.

[50] Nguyen, D.H. and Widrow, B., "Neural networks for self-learning control systems", IEEE Control systems magazine, V. 10, n° 3, (1990), 18–23.

[51] Narendra, K.S., "Adaptive control using neural networks", Neural networks for control, V. 3, (1990), 1–13.

[52] Narendra, K.S. and Mukhopadhyay, S., "Intelligent control using neural networks", IEEE Control systems magazine, V. 12, n° 2, (1992), 11–18.

[53] Narendra, K.S. and Mukhopadhyay, S., "Neural networks in control systems", IEEE Conference on Decision and Control, (1992), 1–6.

[54] Zhang, T. Ge, S.S. and Hang, C.C., "Adaptive neural network control for strict-feedback nonlinear systems using backstepping design", Automatica, V. 36, n° 12, (2000), 1835–1846.

[55] Li, Y. Qiang, S. Zhuang, X. and Kaynak, O., "Robust and adaptive backstepping control for nonlinear systems using RBF neural networks", IEEE Trans Neural Networks, (2004), 693–701.

[56] Wang, D. and Huang, J., "Neural network-based adaptive dynamic surface control for a class of uncertain nonlinear systems in strict-feedback form", IEEE transactions on neural networks, V. 16, n° 1, (2005), 195–202.

[57] Chen, Z. and Jagannathan, S., "Generalized Hamilton–Jacobi–Bellman Formulation - Based Neural Network Control of Affine Nonlinear Discrete-Time Systems", IEEE Transactions on Neural Networks, V. 19, n° 1, (2008), 90–106.

[58] Ge, S.S. Yang, C. and Lee, T.H., "Adaptive Predictive Control Using Neural Network for a Class of Pure-Feedback Systems in Discrete Time", IEEE Transactions on Neural Networks, V. 19, n° 9, (2008), 1599–1614.

[59] Ren, B. Ge, S.S. Tee, K.P. and Lee, T.H., "Adaptive Neural Control for Output Feedback Nonlinear Systems Using a Barrier Lyapunov Function", IEEE Transactions on Neural Networks, V. 21, n° 8, (2010), 1339–1345.

[60] Tong, S.C. Li, Y.M. and Zhang, H., "Adaptive Neural Network Decentralized Backstepping Output-Feedback Control for Nonlinear Large-Scale Systems With Time Delays", IEEE Transactions on Neural Networks, V. 22, n° 7, (2011), 1073–1086.

[61] Rovithakis, G.A. and Christodoulou, M.A., "Adaptive control with recurrent high-order neural networks: theory and industrial applications", Springer Science & Business Media, (2012).

[62] Chen, C.L.P. Liu, Y. and Wen, G., "Fuzzy Neural Network-Based Adaptive Control for a Class of Uncertain Nonlinear Stochastic Systems", IEEE Transactions on Cybernetics , V. 44, n° 5, (2014), 583–93.

[63] Chen, M. Tao, G. and Jiang, B., "Dynamic Surface Control Using Neural Networks for a Class of Uncertain Nonlinear Systems With Input Saturation", IEEE transactions on neural networks and learning systems, V. 26, n° 9, (2015), 2086–2097.

[64] He, W. Chen, Y. and Yin, Z., "Adaptive Neural Network Control of an Uncertain

Robot With Full-State Constraints", IEEE transactions on cybernetics, V. 46, n° 3, (2016), 620–629.

[65] Liu, Y. Li, J. Tong, S. and Chen, C.L.P., "Neural Network Control-Based Adaptive Learning Design for Nonlinear Systems With Full-State Constraints", IEEE transactions on neural networks and learning systems, V. 27, n° 7, (2016), 1562–1571.

[66] Wang, F. Chen, B. Lin, C. Zhang, J. and Meng, X., "Adaptive Neural Network Finite-Time Output Feedback Control of Quantized Nonlinear Systems", IEEE Transactions on Cybernetics, V. 48, n° 6, (2018), 1839–1848.

[67] Sitharthan, R. Parthasarathy, T. Sheeba, R.S. and Ramya, K.C., "An improved radial basis function neural network control strategy-based maximum power point tracking controller for wind power generation system", Transactions of the Institute of Measurement and Control , V. 41, n° 11, (2019), 3158–3170.

[68] Lewis, F.W. Jagannathan, S. and Yesildirak, A., "Neural network control of robot manipulators and non-linear systems", CRC press, (2020).

[69] Lin, C.T. and Lee, C.S.G., "Neural-network-based fuzzy logic control and decision system", IEEE Transactions on computers, V. 40, n° 12, (1991), 1320–1336.

[70] Jang, J.S., "Fuzzy modeling using generalized neural networks and kalman filter algorithm", Association for the Advancement of Artificial Intelligence, V. 91, (1991), p. 762–767.

[71] Jang, J.S. "Rule extraction using generalized neural networks", 4th IFSA World Congr, (1991), 82–86.

[72] Jang, J.S. "ANFIS: adaptive-network-based fuzzy inference system", IEEE transactions on systems, man, and cybernetics, V. 23, n° 3, (1993), 665–685.

[73] Khuntia, S.R. and Panda, S., "Simulation study for automatic generation control of a multi-area power system by ANFIS approach", Applied soft computing, V. 12, n° 1, (2012), 333–341.

[74] García, P. García, C.A. Fernández, L.M. Llorens, F. and Jurado, F., "ANFIS-based control of a grid-connected hybrid system integrating renewable energies", IEEE Transactions on industrial informatics, V. 10, n° 2, (2013), 1107–1117.

[75] Tatikonda, R.C. Battula, V.P. and Kumar, V., "Control of inverted pendulum using adaptive neuro fuzzy inference structure (ANFIS)", IEEE International Symposium on Circuits and Systems, (2010), 1348–1351.

[76] Syahputra, R. and Soesanti, I., "DFIG Control Scheme of Wind Power Using ANFIS Method in Electrical Power Grid System", International Journal of Applied Engineering Research, V. 11, n° 7, (2016), 5256–5262.

[77] Kusagur, A. Kodad, S.F. and Ram, B.V.S., "Modeling, design & simulation of an adaptive neuro-fuzzy inference system (ANFIS) for speed control of induction motor", Journal of Computer Applications, V, 6, n° 12, (2010), 29–44.

[78] Rezakazemi, M. Dashti, A. Asghari, M. and Shirazian, S., "H2-selective mixed matrix membranes modeling using ANFIS, PSO-ANFIS, GA-ANFIS", International Journal of Hydrogen Energy, V. 42, n° 22, (2017), 15211–15225.

[79] Buragohain, M. and Mahanta, C., "A novel approach for ANFIS modelling based on

full factorial design", Applied soft computing, V. 8, n° 1, (2008), 609–625.

[80] Karaboga, D. and Kaya, E., "Adaptive network based fuzzy inference system (ANFIS) training approaches: a comprehensive survey", Artificial Intelligence Review, V. 52, n° 4, (2019), 2263–2293.

[81] Ydstie, B.E., "Extended Horizon Adaptive Control", Proceedings Volumes, V. 17, n° 2, (1985), 911–915.

[82] Keyser, R.M.C. and Van, C.A.R., "Extended Prediction Self-Adaptive Control", IFAC Proceedings Volumes, V. 18, n° 5, (1985), 1255–1260.

[83] Clarke, D.W. Mohtadi, C. and Tuffs, P.S., "Generalized predictive control-Part I. The basic algorithm", Automatica, V. 23, n° 2, (1987), 137–148.

[84] Clarke, D.W. Mohtadi, C. and Tuffs, P.S. "Generalized Predictive Control-Part II Extensions and interpretations", Automatica, V. 23, n° 2, (1987), 149–160.

[85] Kansha, Y. and Chiu, M.S., "Adaptive generalized predictive control based on JITL technique", Journal of Process Control, V. 19, n° 7, (2009), 1067–1072.

[86] Li, Z. and Wang, G., "Generalized predictive control of linear time-varying systems", Journal of the Franklin Institute, V. 354, n° 4, (2017), 1819–1832.

[87] Lucia, S. Tatulea-Codrean, A. Schoppmeyer, C. and Engell, S., "An environment for the efficient testing and implementation of robust NMPC", IEEE Conference on Control Applications (2014), 1843–1848.

[88] Han, H. and Qiao, J., "Nonlinear model-predictive control for industrial processes: An application to wastewater treatment process", IEEE Transactions on Industrial Electronics, V. 61, n° 4, (2014), 1970–1982.

[89] Subramanian, S. Nazari, S. Alvi, M.A. and Engell, S., "Robust NMPC Schemes for the Control of Mobile Robots in the Presence of Dynamic Obstacles", International Conference on Methods & Models in Automation & Robotics, (2018), 768–773.

[90] Mazinan, A.H., "A new algorithm to AI-based predictive control scheme for a distillation column system", The International Journal of Advanced Manufacturing Technology, V. 66, n° 9, (2013), 1379–1388.

[91] Lu, Q. Shi, P. Lam, H.K. and Zhao, Y., "Interval Type-2 Fuzzy Model Predictive Control of Nonlinear Networked Control Systems", IEEE Transactions on Fuzzy systems, V. 23, n° 6, (2015), 2317–2328.

[92] Thangavel, S. Lucia, S. Paulen, R. and Engell, S., "Dual robust nonlinear model predictive control: A multi-stage approach", Journal of Process Control, V. 72, (2018), 39–51.

[93] Maner, B.R. Doyle, F.J. Ogunnaike, B.A. and Pearson, R.K. "Nonlinear model predictive control scheme using second order volterra models", American Control Conference, (1994), 3253–3257.

[94] Gruber, J.K. Ramirez, D.R. Alamo, T. and Bordons, C., "Nonlinear min-max model predictive control based on volterra models application to a pilot plant", European Control Conference, (2009), 1112–1117.

[95] Díaz-Mendoza, R. and Budman, H., "Robust nonlinear model predictive control using

volterra models and the structured singular value (μ)", IFAC Proceedings Volumes, V. 42, n° 11, (2009), 375–380.

[96] Ait-Sahed, O. Kara, K. and Benyoucef, A., "Artificial bee colony-based predictive control for non-linear systems", Transactions of the Institute of Measurement and Control, V. 37, n° 6, (2015), 780–792.

[97] Ait-Sahed, O. Kara, K. Benyoucef, A. and Hadjili, M.L., "An efficient artificial bee colony algorithm with application to nonlinear predictive control", International Journal of General Systems, V. 45, n° 4, (2016), 393–417.

[98] Huang, Y.L. Lou, H.H. Gong, J.P. and Edgar, T.F., "Fuzzy model predictive control", IEEE Transactions on Fuzzy Systems, V. 8, n° 6, (2000), 665–678.

[99] Han, H.G. Wu, X.L. and Qiao, J.F., "Real-time model predictive control using a self-organizing neural network", IEEE transactions on neural networks and learning systems, V. 24, n° 9, (2013), 1425–36.

[100] Patan, K., "Two stage neural network modelling for robust model predictive control", ISA transactions, V. 72, (2018), 56–65.

[101] Soloway, D. and Haley, P.J., "Neural generalized predictive control a Newton-Raphson implementation", IEEE International Symposium on Intelligent Control, (1996), 277–282.

[102] Liu, G.P. Kadirkamanathan, V. and Billings, S.A., "Predictive control for non-linear systems using neural networks", International Journal of Control, V. 71, n° 6, (1998), 1119–1132.

[103] Botto, M.A. Van Den Boom, T.J.J. Krijgsman, A. and Da Costa, J.S., "Predictive control based on neural network models with I/O feedback linearization", International Journal of Control, V. 72, n° 17, (1999), 1538–1554.

[104] Sarimveis, H. and Bafas, G., "Fuzzy model predictive control of non-linear processes using genetic algorithms", Fuzzy sets and systems, V. 139, n° 1, (2003),59–80.

[105] Li, Y. Shen, J. Lee, K.Y. and Liu X., "Offset-free fuzzy model predictive control of a boiler-turbine system based on genetic algorithm", Simulation modelling practice and theory, V. 26, (2012), 77–95.

[106] Coelho, J.P. De Moura Oliveira, P.B. and Boaventura Cunha, J., "Greenhouse air temperature control using the particle swarm optimisation algorithm", IFAC Proceedings Volumes, V. 35, n° 1, (2002), 43–47.

[107] Zhixiang, H. Hui, C. and Heqing, L., "Neural networks predictive control using AEPSO", Chinese Control Conference, (2008), 180–183.

[108] Zimmer, A. Schmidt, A, Ostfeld, A. and Minsker, B., "Evolutionary algorithm enhancement for model predictive control and real-time decision support. Environmental Modelling & Software, V. 69, (2015), 330–341.

[109] Rao, R.V. Savsani, V.J. and Vakharia, D.P., "Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems", Computer-Aided Design, V. 43, n° 3, (2011), 303–315.

[110] Rao, R.V. Savsani, V.J. and Vakharia, D.P., "Teaching-Learning-Based Optimization: An optimization method for continuous non-linear large scale

problems", Information sciences, V. 183, n° 1, (2012), 1–15.

[111] Levin, E. Gewirtzman, R. and Inbar, G.F., "Neural network architecture for adaptive system modeling and control". Neural Networks, V. 4, n° 2, (1991), 185–191.

[112] Kuperstein, M., "Neural model of adaptive hand-eye coordination for single postures", Science, V. 239, (1988), 1308–1311.

[113] Daosud, W. Thitiyasook, P. and Arpornwichanop, A. and Kittisupakorn, P. and Hussain, M.A., "Neural network inverse model-based controller for the control of a steel pickling process", Computers & Chemical Engineering, V. 29, n° 10, (2005), 2110–2119.

[114] Plett, G.L., "Adaptive inverse control of linear and nonlinear systems using dynamic neural networks", IEEE transactions on neural networks, V. 14, n° 2, (2003), 360–376.

[115] Gupta, M.M., "Neuro-control systems: theory and applications", IEEE press, (1994).

[116] Lee, C.S.G. and Lin, C.T., "Supervised and unsupervised learning with fuzzy similarity for neural-network-based fuzzy logic control systems", IEEE International Conference on Systems, Man, and Cybernetics, (1992), 688–693.

[117] Nørgård, P.M. Ravn, O. Poulsen, N.K. and Hansen, L.K., "Neural networks for modelling and control of dynamic systems-A practitioner's handbook", (2000).

[118] Kumpati, S.N. and Kannan, P., "Identification and control of dynamical systems using neural networks", IEEE Transactions on neural networks, V. 1, n° 1, (1990), 4–27.

[119] Narendra, K.S. "Adaptive control using neural networks", Neural Networks Control, V. 3, (1991).

[120] Velagic, J. Osmic, N. and Lacevic, B., "Design of neural network mobile robot motion controller", New Trends in Technologies. IntechOpen, (2010).

[121] Ch, S.B. kumar, S.V. and Dr, D.V.A. "Neural Network Controller for Enhancement of Uninterruptible Power Supply Inverter". International Journal of Recent Technology, V. 1, (2012), 10–6.

[122] Iiguni, Y. Sakai, H. and Tokumaru, H., "A nonlinear regulator design in the presence of system uncertainties using multilayered neural network", IEEE transactions on neural networks, V. 2, n° 4, (1991), 410–417.

[123] Badr, A.Z., "Neural Network Based Adaptive PID Controller", IFAC Proceedings Volumes, V. 30, n° 6, (1997), 251–257.

[124] Benrabah, M. Kara, K. AitSahed, O. and Hadjili, L., "Adaptive Neural Network PID Controller", International Conference on Environment and Electrical Engineering, (2019), 1–6.

[125] Togai, M., "Application of fuzzy inverse relations to synthesis of a fuzzy controller for dynamic systems", IEEE Conference on Decision and Control, (1984), 904–905.

[126] Lee, C.C., "Fuzzy logic in control systems: fuzzy logic controller", IEEE Transactions on systems, V. 20, n° 2, (1990), 404–418.

[127] Wang, L.X., "Stable adaptive fuzzy control of nonlinear systems", IEEE Transactions

on fuzzy systems, V. 1, n° 2, (1993), 146–155.

[128] Chen, B. Liu, X. Liu, K. and Lin, C., "Direct adaptive fuzzy control of nonlinear strict-feedback systems". Automatica, V. 45, n° 6, (2009), 1530–1535.

[129] Moore, C.G. and Harris, C.J., "Indirect adaptive fuzzy control", International Journal of Control, V. 56, n° 2, (1992), 441–468.

[130] Carvajal, J. Chen, G. and Ogmen, H., "Fuzzy PID controller: Design, performance evaluation, and stability analysis", Information sciences, V. 123, n° 3, (2000), 249–270.

[131] Tang, K.S. Man, K.F. Chen, G. and Kwong, S., "An optimal fuzzy PID controller". IEEE transactions on industrial electronics, V. 48, n° 4, (2001), 757–765.

[132] Kim, J.H. Park, J.H. Lee, S.W. and Chong, E.K.P., "A two-layered fuzzy logic controller for systems with deadzones", IEEE Transactions on Industrial Electronics, V. 41, n° 2, (1994), 155–162.

[133] Lughofer, E., "Evolving fuzzy systems-methodologies, advanced concepts and applications", Springer, Berlin, (2011).

[134] Kasabov, N.K., "Evolving connectionist systems: the knowledge engineering approach", Springer Science & Business Media, (2007).

[135] Halgamuge, S.K. and Glesner, M., "Neural networks in designing fuzzy systems for real world applications", Fuzzy sets and systems, V. 65, n° 1, (1994), 1–12.

[136] Tschichold-Gürman, N., "RuleNet-a new knowledge-based artificial neural network model with application examples in robotics", ETH Zürich, PhD Thesis, (1996).

[137] Berenji, H.R. and Khedkar, P., "Learning and tuning fuzzy logic controllers through reinforcements", IEEE Transactions on neural networks, V. 3, n° 5, (1992), 724–740.

[138] Nauck, D. Klawonn, F. and Kruse, R., "Foundations of neuro-fuzzy systems", John Wiley & Sons, (1997).

[139] Singh M, Chandra A. Real-time implementation of ANFIS control for renewable interfacing inverter in 3P4W distribution network. IEEE Trans Ind Electron 2012;60:121–8.

[140] Denai, M.A. Palis, F. and Zeghbib, A., "ANFIS based modelling and control of non-linear systems: a tutorial", IEEE International Conference on Systems, Man and Cybernetics, (2004), 3433–3438.

[141] Abu-Rub, H. Iqbal, A. Ahmed, S.M. Peng, F.Z. Li, Y. and Baoming, G., "Quasi-Z-source inverter-based photovoltaic generation system with maximum power tracking control using ANFIS", IEEE Transactions on Sustainable Energy, V. 4, n° 1, (2012), 11–20.

[142] Chang, B.C.H. and Halgamuge, S., "Model free online adaptive feedback control with FuNe I AFC neuro-fuzzy system", International Conference on Fuzzy Systems , (2000), 997–1000.

[143] Chang, B. and Halgamuge, S., "Rule Based Reinforcement Learning Algorithm for Weight Update in FuNe I Adaptive Feedback Controller", (2005).

[144] Nürnberger, A. Nauck, D. and Kruse, R., "Neuro-fuzzy control based on the NEFCON-model: recent developments", Soft Computing, V. 2, n° 4, (1999), 168–182.

[145] Song, Y. Chen, Z. and Yuan, Z., "New chaotic PSO-based neural network predictive control for nonlinear process", IEEE transactions on neural networks, V. 18, n° 2, (2007), 595–600.

[146] Wysocki, A. and Ławryńczuk, M., "Elman neural network for modeling and predictive control of delayed dynamic systems", Archives of Control Sciences, V. 26, (2016).

[147] Espinosa, O. Jose, J. Vandewalle, J. and Wertz, V., "Fuzzy Logic, Identification, and Predictive Control", Springer, London, (2005).

[148] Han, H.G. Liu, Z. and Qiao, J.F., "Fuzzy neural network-based model predictive control for dissolved oxygen concentration of WWTPs", International Journal of Fuzzy Systems, V. 21, n° 5, (2019), 1497–1510.

[149] Lu, C.H. and Tsai, C.C., "Generalized predictive control using recurrent fuzzy neural networks for industrial processes", Journal of process control, V. 17, n° 1, (2007), 83–92.

[150] Lu, C.H., "Wavelet fuzzy neural networks for identification and predictive control of dynamic systems", IEEE Transactions on Industrial Electronics, V. 58, n° 7, (2010), 3046–3058.

[151] Abedi, M. Yaghoobi, M. and Raesian, N., "Optimization of model predictive controller parameters based on Imperialist Competitive algorithm", Iranian Conference on Electrical Engineering, (2013), 1–5.

[152] Davtyan, A. Hoffmann, S. and Scheuring, R., "Optimization of model predictive control by means of sequential parameter optimization", Computational Intelligence in Control and Automation, (2011), 11–16.

[153] Polya, G., "How to solve it ", Princeton University, New Jersey, (1945).

[154] Glover, F., "Future paths for integer programming and links to artificial intelligence", Computers & operations research, V. 13, n° 5, (1986), 533–549.

[155] Osman, I.H. and Kelly, J.P., "Meta-heuristics: an overview", Meta-Heuristics, V. 1, n° 21, (1996), 1–21.

[156] Kirkpatrick, S. Gelatt, C.D. and Vecchi, M.P., "Optimization by simulated annealing" Science, V. 220, (1983), 671–680.

[157] Mladenović, N. and Hansen, P., "Variable neighborhood search", Computers & operations research, V. 24, n° 11, (1997), 1097–1100.

[158] Feo, T.A. and Resende, M.G.C., "A probabilistic heuristic for a computationally difficult set covering problem", Operations research letters, V. 8, n° 2, (1989), 67–71.

[159] Dorigo, M. and Di Caro, G., "Ant colony optimization: a new meta-heuristic", Proceedings of the 1999 congress on evolutionary computation, (1999), 1470–1477.

[160] Kennedy, J. and Eberhart, R., "Particle swarm optimization", international conference on neural networks, (1995), 1942–1948.

[161] Karaboga, D., "An idea based on honey bee swarm for numerical optimization", Technical report-tr06, (2005).

[162] Vajda, S., "Mathematical programming", Courier Corporation, (2009).

[163] Jaffar, J. and Lassez, J.L., "Constraint logic programming", symposium on Principles of programming languages, (1987), 111–119.

[164] Maaranen, H. Miettinen, K. and Penttinen, A., "On initial populations of a genetic algorithm for continuous optimization problems", Journal of Global Optimization , V. 37, n° 3, (2007), 405.

[165] Maaranen, H. Miettinen, K. and Mäkelä, M.M., "Quasi-random initial population for genetic algorithms", Computers & Mathematics with Applications, V. 47, n° 12, (2004), 1885–1895.

[166] Sacco, W.F. and Rios-Coelho, A.C.. "On initial populations of differential evolution for practical optimization problems", Computational Intelligence, Optimization and Inverse Problems with Applications in Engineering, Springer, (2019), 53–62.

[167] Talbi, E.G., "Metaheuristics: from design to implementation", John Wiley & Sons, (2009).

[168] Clerc, M., "Particle swarm optimization". John Wiley & Sons, (2010).

[169] Storn, R. and Price, K., "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces", Journal of global optimization, V. 11, n° 4, (1997), 341–359.

[170] Michalewicz, Z., "Genetic algorithms+ data structures= evolution programs", Springer Science & Business Media, (2013).

[171] Iba, H. and Aranha, C.C., "Practical Applications of Evolutionary Computation to Financial Engineering", Springer, (2012).

[172] Wolpert, D.H. and Macready, W.G., "No free lunch theorems for search", Technical Report, Santa Fe Institute, (1995).

[173] Wolpert, D.H. and Macready, W.G., "No free lunch theorems for optimization", IEEE transactions on evolutionary computation, V. 1, n° 1, (1997), 67–82.

[174] Eberhart, R. and Kennedy, J., "A new optimizer using particle swarm theory", International Symposium on Micro Machine and Human Science, (1995), 39–43.

[175] Xinchao, Z. "A perturbed particle swarm algorithm for numerical optimization", Applied Soft Computing, V. 10, n° 1, (2010), 119–24.

[176] Arani, B.O. Mirzabeygi, P. and Panahi, M.S., "An improved PSO algorithm with a territorial diversity-preserving scheme and enhanced exploration–exploitation balance", Swarm and Evolutionary Computation, V. 11, (2013), 1–15.

[177] Shi, Y. and Eberhart, R., "A modified particle swarm optimizer", IEEE international conference on evolutionary computation proceedings, (1998), 69–73.

[178] Shi, Y. and Eberhart, R.C., "Fuzzy adaptive particle swarm optimization". Proceedings of the 2001 congress on evolutionary computation, (2001), 101–106.

[179] Liang, J.J. Qin, A.K. Suganthan, P.N. and Baskar, S., "Comprehensive learning

particle swarm optimizer for global optimization of multimodal functions", IEEE transactions on evolutionary computation, V. 10, n° 3, (2006), 281–295.

[180] Chen, S. Montgomery, J. and Bolufé-Röhler, A., "Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution", Applied Intelligence, V. 42, n° 3, (2015), 514–26.

[181] Bolufé-Röhler, A. and Chen, S., "Minimum population search-lessons from building a heuristic technique with two population members", IEEE Congress on Evolutionary Computation, (2013), 2061–2068.

[182] Rao, R.V. and Patel, V., "An improved teaching-learning-based optimization algorithm for solving unconstrained optimization problems", Scientia Iranica , V. 20, n° 3, (2013), 710–720.

[183] Åström, K.j. and Hägglund, T., "PID controllers : theory, design, and tuning", Research Triangle Park, NC: Instrument society of America, (1995).

[184] Yu, W., "PID Control with Intelligent Compensation for Exoskeleton Robots", Academic Press, (2018).

[185] Garrido, R. and Trujano, M.A., "Stability analysis of a visual pid controller applied to a planar parallel robot", International Journal of Control, Automation and Systems , V. 17, n° 6, (2019), 1589–1598.

[186] Duong, P.L.T. and Lee, M., "Design of robust PID controller for processes with stochastic uncertainties", Computer Aided Chemical Engineering, V. 29, (2011), 512–516.

[187] Memon, F. and Shao, C., "An optimal approach to online tuning method for PID type iterative learning control", Journal of Control, Automation and Systems (2020), 1–10.

[188] Tepljakov, A. Alagoz, B.B. Yeroglu, C. Gonzalez, E. HosseinNia, H.S. and Petlenkov, E., "FOPID Controllers and Their Industrial Applications: A Survey of Recent Results". IFAC-PapersOnLine, V. 51, n° 4, (2018), 25–30.

[189] Shah, P. Agashe, S., "Review of fractional PID controller", Mechatronics, V. 38, (2016), 29–41.

[190] Su, Y.X. Sun, D. and Duan, B.Y., "Design of an enhanced nonlinear PID controller", Mechatronics, V. 15, n° 8, (2005), 1005–1024.

[191] Chen, W.H. Balance, D.J. Gawthrop, P.J. Gribble. J.J. and O'Reilly, J., "Nonlinear PID predictive controller", IEE Proceedings-Control Theory and Applications, V. 146, n° 6, (1999), 603–611.

[192] Girgis, M.E. Fahmy, R.A. and Badr, R.I., "Optimal fractional-order PID control for plasma shape, position, and current in Tokamaks", Fusion Engineering and Design, V. 150, (2020), 1–7.

[193] Pradhan, R. Majhi, S.K. Pradhan, J.K. and Pati, B.B., "Optimal fractional order PID controller design using Ant Lion Optimizer", Ain Shams Engineering Journal, V. 11, n° 2, (2019), 1–11.

[194] Kang, J. Meng, W. Abraham, A. and Liu, H., "An adaptive PID neural network for complex nonlinear system control", Neurocomputing, V. 135, (2014), 79–85.

[195] Huailin, S., "Analysis of PID neural network multivariable control systems", Acta Automatica Sinica, V. 25, n° 1, (1999), 105–11.

[196] Yongquan, Y. Ying, H. and Bi, Z., "A PID neural network controller", International Joint Conference on Neural Networks, (2003), 1933–1938.

[197] Zribi, A. Chtourou, M. and Djemel, M., "A new PID neural network controller design for nonlinear processes", Journal of Circuits, Systems and Computers, V. 27, n° 4, (2018), 1850065.

[198] Asgharnia, A. Shahnazi, R. and Jamali, A., "Performance and robustness of optimal fractional fuzzy PID controllers for pitch control of a wind turbine using chaotic optimization algorithms" ISA Transactions, V. 79, (2018), 27–44.

[199] Somwanshi, D. Bundele, M. Kumar, G. and Parashar, G., "Comparison of Fuzzy-PID and PID Controller for Speed Control of DC Motor using LabVIEW". Procedia Computer Science, V. 152, (2019), 252–260.

[200] Raj, R. et Mohan, B.M., "Takagi-Sugeno fuzzy PID controllers: mathematical models and stability analysis with multiple fuzzy sets", International Journal of Fuzzy Computation and Modelling, V. 3, n° 1, (2020), 33–60.

[201] Kudinov, Y.I., Kolesnikov, V.A., Pashchenko, F.F., Pashchenko, A.F. and Papic, L., "Optimization of Fuzzy PID Controller’s Parameters", Procedia Computer Science, V. 103, (2017), 618–620.

[202] Kumar, A. and Kumar, V., "A novel interval type-2 fractional order fuzzy PID controller: Design, performance evaluation, and its optimal time domain tuning", ISA transactions, V. 68, (2017), 251–275.

[203] Rugh, W.J., "Design of nonlinear PID controllers", AIChE Journal, V. 33, n° 10, (1987), 1738–1742

[204] Scherer, R., Kalla, S.L., Tang, Y. and Huang, J., "The Grünwald–Letnikov method for fractional differential equations", Computers & Mathematics with Applications, V. 62, n° 3, (2011), 902–917.

[205] Lund, M.R., "Intégrales de Riemann—Liouville et potentiels", Acta Litt Ac Sci Regiae Univ Hungaricae Fr AM Kir Ferencz József-Tudományegyetem Tudományos Közleményei Math Tudományok Sect Sci Math, (1938), 9-1.

[206] Abdeljawad, T., "On Riemann and Caputo fractional differences". Computers & Mathematics with Applications, V. 62, n° 3, (2011), 1602-1611.

[207] Garrappa, R., Kaslik, E., and Popolizio, M., "Evaluation of fractional integrals and derivatives of elementary functions: Overview and tutorial", Mathematics, V. 7, n° 5, (2019), 407.

[208] Fang, H., Chen, L. and Shen, Z., "Application of an improved PSO algorithm to optimal tuning of PID gains for water turbine governor", Energy Conversion and Management, V. 52, n°4, (2011), 1763-1770.

[209] Wang, J., An, D. and Lou, C., "Application of fuzzy-PID controller in heating ventilating and air-conditioning system", International Conference on Mechatronics and Automation, (2006), 2217–2222.

[210] Yu, W. and Rosen, J., "Neural PID control of robot manipulators with application to

an upper limb exoskeleton", IEEE Transactions on cybernetics, V 43, n° 2, (2013), 673-684.

[211] Qi, Y. and Meng, Q., "The application of fuzzy PID control in pitch wind turbine", Energy Procedia, V. 16, (2012), 1635-1641.

[212] Shu, H. and Pi, Y., "PID neural networks for time-delay systems", Computers & Chemical Engineering, V. 24, n° 2, (2000), 859-862.

[213] krishna, S. and Vau, S., "Fuzzy PID based adaptive control on industrial robot system". Materials Today: Proceedings, V. 5, n° 5, (2018), 13055-13060.

[214] Ulpiani, G., Borgognoni, M., Romagnoli, A. and Perna, C.Di., "Comparing the performance of on/off, PID and fuzzy controllers applied to the heating system of an energy-efficient building", Energy and Buildings, V. 116, (2016), 1-17.

[215] Chiou, J-S., Tsai, S-H. and Liu, M-T., "A PSO-based adaptive fuzzy PID-controllers", Simulation Modelling Practice and Theory, V. 26, (2012), 49-59.

[216] Valluru, S.K. and Singh, M., "Performance investigations of APSO tuned linear and nonlinear PID controllers for a nonlinear dynamical system", Journal of Electrical Systems and Information Technology, V. 5, n° 3, (2018), 442-452.

[217] Solihin, M.I., Tack, L.F. and Kean M.L., "Tuning of PID controller using particle swarm optimization (PSO)", Proceeding of the International Conference on Advanced Science, Engineering and Information Technology, V. 1, (2011), 458-461.

[218] Das, K.R., Das, D. and Das, J., "Optimal tuning of PID controller using GWO algorithm for speed control in DC motor", International Conference on Soft Computing Techniques and Implementations (ICSCTI), (2015), 108-112.

[219] Mishra, A.K., Das, S.R., Ray, P.K., Mallick, R.K., Mohanty, A. and Mishra, D,K., "PSO-GWO Optimized Fractional Order PID Based Hybrid Shunt Active Power Filter for Power Quality Improvements", IEEE Access, V. 8, (2020), 74497-74512.

[220] Sahu, R.K., Panda, S., Rout, U.K. and Sahoo, D.K., "Teaching learning based optimization algorithm for automatic generation control of power system using 2-DOF PID controller", International Journal of Electrical Power & Energy Systems, V. 77, (2016), 287-301.

[221] Chatterjee, S. and Mukherjee, V., "PID controller for automatic voltage regulator using teaching–learning based optimization technique", International Journal of Electrical Power & Energy Systems, V. 77, (2016), 418-429.

[222] Senberber, H. and Bagis, A., "Fractional PID controller design for fractional order systems using ABC algorithm", Electronics IEEE, (2007), 1-7.

[223] Kaliappan, V. and Thathan, M., "Enhanced ABC based PID controller for nonlinear control systems", Indian Journal of Science and Technology, V. 8, (2015), 48.

[224] Ko, C-N. and Wu, C-J., "A PSO-tuning method for design of fuzzy PID controllers" Journal of Vibration and Control, V. 14, n° 3, (2008), 375-395.

[225] Chavoshian, M., Taghizadeh, M. and Mazare, M., "Hybrid dynamic neural network and PID control of pneumatic artificial muscle using the PSO algorithm", International Journal of Automation and Computing, V. 17, n° 3, (2020), 428-438

[226] Dong, X., Zhang, Z. and Chen, C., "Applying genetic algorithm to on-line updated PID neural network controllers for ball and plate system". Fourth International Conference on Innovative Computing, Information and Control (ICICIC), (2009), 751-755.

[227] Jones, A.H., "Genetic tuning of neural non-linear PID controllers. Artif. Neural Nets Genetic Algorithms", Springer, (1995), 412–415.

[228] Benrabah, M., Kara, K., AitSahed, O. and Hadjili, M.L., "Adaptive Fourier Series Neural Network PID Controller", International Journal Control Automation System, (2021).

[229] Li, C., Lili, T., Hui, C., Qi, R. and Feng, Q., "Multivariable generalized predictive control based on receding feedback correction in binary distillation process", Proceedings of the 10th World Congress on Intelligent Control and Automation, (2012), 1098-1102.

[230] Pugh, G.A,. "Synthetic neural networks for process control", Computers & industrial engineering, V. 17, n° 1, (1989), 24-26.

[231] Karlin, S., "The structure of dynamic programing models", Naval Research Logistics Quarterly, V. 2, n° 4, (1955), 285-294.

[232] Song, Z., Hofmann, H., Li, J., Han, X. and Ouyang, M., "Optimization for a hybrid energy storage system in electric vehicles using dynamic programing approach", Applied Energy, V. 139, (2015), 151-162.

[233] Crampin, M., "On the differential geometry of the Euler-Lagrange equations, and the inverse problem of Lagrangian dynamics", Journal of Physics A: Mathematical and General, V. 14, n° 10, (1981), 2567.

[234] Park, J. and Chung, W.K., "Analytic nonlinear H/sub/spl infin//inverse-optimal control for Euler-Lagrange system", IEEE Transactions on Robotics and Automation, V. 16, n° 6, (2000), 847-854

[235] Betts, J.T., "Survey of numerical methods for trajectory optimization", Journal of guidance, control, and dynamics, V. 21, n° 2, (1998), 193-207.

[236] Levin, A.U. and Narendra, K.S., "Identification of nonlinear dynamical systems using neural networks", Neural systems for control. Academic Press, (1997), 129-160.

[237] Setiono, R. and Hui, L.C.K., "Use of a quasi-Newton method in a feedforward neural network construction algorithm", IEEE Transactions on Neural Networks, V. 6, n° 1, (1995), 273-277.

[238] Moré, J.J., "The Levenberg-Marquardt algorithm: implementation and theory", Numerical analysis. Springer, Berlin, Heidelberg, (1978), 105-116.

[239] Leung, H. and Haykin, S., "The complex backpropagation algorithm", IEEE Transactions on signal processing, V. 139, n° 9, (1991), 2101–2104.

[240] Lin, F., "Robust Control Design: An Optimal Control Approach", John Wiley & Sons, (2007).