INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9226879

Intelligent coordination of multiple systems with neural networks

Cui, Xianzhong, Ph.D.

The University of Michigan, 1992

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

INTELLIGENT COORDINATION OF MULTIPLE SYSTEMS WITH NEURAL NETWORKS

by

Xianzhong Cui

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering: Systems)
in The University of Michigan
1992

Doctoral Committee:

Professor Kang G. Shin, Chairperson
Professor Ramesh C. Jain
Professor N.Harris McClamroch
Associate Professor Michael W. Walker

To My Wife and All Those Who Helped Me.

ACKNOWLEDGEMENTS

My sincere thanks go to my advisor Prof. Kang G. Shin. His thoughtful guidance, constant encouragement and kind support are the bases for my completing this dissertation. I always take his character and working attitude as a model for judging my own progress. I would also like to thank the members of my dissertation committee — Prof. Ramesh C. Jain, Prof. N. Harris McClamroch and Prof. Michael W. Walker, for their valuable time given without reservation in helping me and reviewing this work.

Throughout the years of studying at the University of Michigan, I have always been encouraged and consoled by my wife, parents, parents in–law, sister and brother, even though they have faced many difficulties. I hope that their deep love and heartfelt devotion could be requited a little with the completion of my Ph.D. degree. Thanks to Mr. Harold Ludwig and Mrs. Winnie Ludwig — my host family — for their solicitude and prayers.

I would like to express special gratitude to Mr. Li Xiwu (a senior engineer of the Electric Power Research Institute, P. R. China), Ms. Liu Xiuying (the former consul of the Consulate General of P. R. China in Chicago), and Mr. Ma Songjiang (my classmate in Beijing Graduate School, North China Institute of Electric Power) for their help in my family reunion and my study in the U. S. I thank the Government of P. R. China, which supplied my living expenses from September 1986 to August 1987, when I was a visiting researcher at the University of Michigan.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

ABSTRACT

INTELLIGENT COORDINATION OF MULTIPLE SYSTEMS WITH NEURAL NETWORKS

by
Xianzhong Cui

Chairperson: Kang G. Shin

Many control applications require cooperation of two or more independently designed, separately located, but mutually affecting, subsystems. In addition to the proper functioning of each subsystem, their effective coordination is very important in order to achieve the desired performance. This has led to the development of new multiple–system coordinators and the evaluation of their performance in real–time implementation. Two coordination schemes have been proposed: a knowledge–based coordinator (KBC) and a neural network–based coordinator (NNBC). Either of them functions as a high–level coordinator in a hierarchical system. In such a hierarchical structure, the detailed structure and/or parameters of low–level subsystems need not be known to the coordinator, so that each subsystem can be designed separately.

The basic idea of the KBC is to estimate the effects of commands to low–level subsystems using a predictor and to modify them by searching a knowledge base in order to achieve the desired performance. A general–purpose predictor has been designed for MIMO (multiple–input, multiple–output) systems using neural networks (NNs). By introducing the predictor, the knowledge base for multiple–system coordination

is greatly simplified, and each command is evaluated before its execution.

The basic structure of the NNBC is a multilayer perceptron. NNs are usually trained by using the output errors at its OUTPUT layer. However, when an NN is used to control a plant directly, these errors are unknown, since the desired control actions are unknown. This implies that the conventional back propagation training algorithm cannot be applied to control problems directly. A simple training algorithm has been developed which enables the NNBC to be trained by using the output errors of the controlled plant, thus enhancing the NN's ability to handle control applications.

The effects of computing time delay on the stability and performance of controlled systems have been carefully studied both for general robot control systems and for the proposed coordinators. For the qualitative analysis of the computing time delay, a generic criterion is derived in terms of system stability. For quantitative analysis, upper bounds of computing time delay on system stability and performance are derived for a robot control system. These upper bounds can be used as extra constraints on controller design and selection of CPUs used to implement the control algorithm.

The coordination of two robots holding an object, the coordination of multiple robots to avoid collision, and the control of a nonlinear thermo–process are investigated to test the capability of the proposed schemes. Because the internal structure and parameters of the low–level subsystems are not affected by using either the KBC or the NNBC, some commercially–designed servo controllers can be coordinated to accomplish more sophisticated tasks than originally intended.

CHAPTER I

INTRODUCTION

1.1 Motivation and Purpose

Many control applications require cooperation of two or more independently designed, separately located, but mutually affecting, subsystems. Examples of these applications are the coordinated control of multiple robots holding a single object, coordination of multiple robots working in a common workspace to avoid collision, coordinated control of a main steam temperature and a reheater steam temperature in a thermal power plant, and coordination of multiple generating units for economical load distribution in an electric power system. The coordination problem forms a hierarchical structure in which the internal structure and parameters of low–level subsystems should not be affected by adding a high–level coordinator. In addition to the proper functioning of each subsystem, effective coordination of all subsystems is very important in order to achieve the desired performance. Multiple–system coordination can in general be stated as a constrained optimization problem. However, solving the coordination problem is very difficult due mainly to the lack of a precise system model and/or dynamic parameters, as well as the lack of efficient tools for system analysis, design, and real–time computation of optimal solutions. Therefore, new methods for design and analysis are needed to achieve the closed–loop coordi-

nation of multiple systems. The goals of this research are to develop practical and general design methods for multiple–system coordination with a hierarchical structure, and to evaluate the performance of their real–time implementation.

Although some basic principles in coordinating multiple systems were developed in early 80s [LMO82], most related publications addressed only conceptual interpretation, and very few of them dealt with actual applications. In recent years, certain special coordination problems have drawn considerable attention. One of the challenging topics is coordinated control of multiple robots. Assuming complete knowledge about robot dynamics, most published results are specially designed for the purpose of coordinating multiple robots. However, most of them are not based on a hierarchical structure and are usually not suitable for coordinating two robots which are built with independently–designed, commercial servo controllers.

In this dissertation, we focus on the coordinated control of several physically distributed systems. Generally, there are four levels of hierarchy in a coordination system, as shown in Fig. 1.1. The highest level is the *monitoring and interface level*, which provides man–machine interface, goal setting, and decision making. The next level is the *planning and supervision level* in which the commands to conduct some general, descriptive tasks are transformed to a specified command sequence while satisfying a set of constraints. The third level is called the *coordination level* and is designed to modify the commands from the higher level, so that the command sequence becomes executable without violating the constraints under environmental changes. The lowest level is the *servo control level.* In this level, each subsystem is equipped with a servo controller which is designed separately from and independently of others. The servo controllers are distributed physically and coupled whenever they need to perform cooperative tasks. These subsystems are said to be *loosely–coupled* if

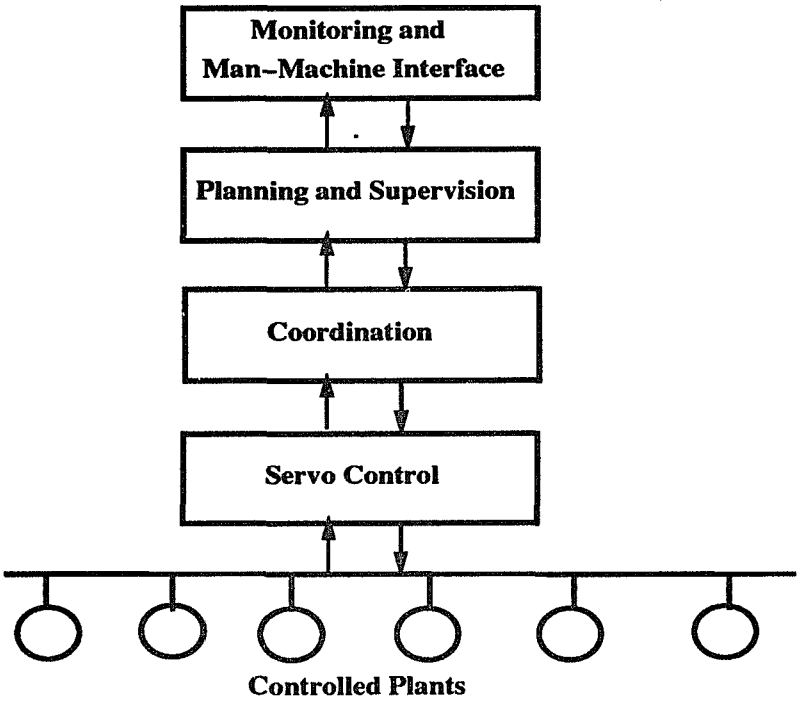


Figure 1.1: Conceptual structure of a hierarchical coordinated control system.

the internal structure and parameters of the subsystems do not affect each other, but must satisfy some common external constraints. For example, multiple robots work in a common workspace and are coordinated to avoid collision. Otherwise, these subsystems are *tightly–coupled.* For example, multiple robots hold a single object rigidly. Obviously, tightly–coupled subsystems are more difficult to coordinate than loosely–coupled ones.

In a hierarchical structure, the higher the level is, the more intelligence it has for decision making, but the less precision of knowledge it knows about lower levels. Each level should be independent of the others in the sense that the internal structure and parameters will not be affected by adding other levels. Based on these principles, two different schemes for multiple system coordination have been developed in this dissertation by using the techniques of intelligent control and neural networks (NNs). In the first scheme, a knowledge–based coordinator is designed by combining the

techniques of intelligent control and neural networks. From the viewpoint of a high level, in order not to interfere in the internal structure/parameters of the low level, the only control action to take is to issue a sequence of appropriate commands to low–level subsystems. Our basic idea is to estimate the effects of these commands using a predictor and then modify the commands through a search in a knowledge base in order to achieve the desired performance. In the second scheme, an NN–based coordinator is developed. One of the main properties of neural networks for control applications is that they can be used to approximate any continuous mapping. Therefore, a multiple–system can be coordinated directly using neural networks, provided we can train such a neural network to learn the relationship between the corresponding responses of the low–level subsystems and the appropriate high–level coordination commands. Because the internal structure and parameters of the lower levels are not affected by using either of the proposed methods, it is possible that some commercially–designed servo controllers could be coordinated to perform more sophisticated tasks than originally intended.

To implement the proposed methods on digital computers, all computations must be finished in real–time. Hence, our research also deals with the real–time computation of coordination commands. Traditionally, the performance of a control system is evaluated by such criteria as stability, rise time, maximum overshoot, and so on. In addition to these, for a real–time control system, the most important issue is timeliness and reliability. A controller computer implements the control algorithms by executing a sequence of instructions. Unlike analog control systems, the reliability of a digital control system depends not only on the MTBF (mean time between failures) of the controller hardware and software, but also on the delay in executing control algorithms on the controller computer. The execution time, or the computing

time delay, for a control algorithm is defined as the period from its trigger to the generation of a corresponding control command. It is an extra time delay introduced into the feedback loop in a controlled system. Because of the existence of conditional branches, resource sharing delays, and processing exceptions, the computing time delay for a given control algorithm is usually a continuous, random variable which is smaller than the sampling interval. This extra time delay cannot be treated as a constant *a priori* and added into controller design. Thus, it is very important to analyze the effects of computing time delay on control system performance when control algorithms are implemented on digital computers. This is especially true for the control/coordination systems with heuristic search and/or symbolic reasoning.

1.2 Overview of the Research

Following an extensive survey of multiple–system coordination, intelligent control, and neural networks in Chapter 2, we propose a hierarchical knowledge–based controller for a single system in Chapter 3. As a starting point in developing a knowledge–based coordinator, we investigate the function and structure of the knowledge–based controller. A knowledge base is designed for which the knowledge acquisition and representation are analyzed. The inference process will conduct a goal–oriented search in the knowledge base. Since a knowledge–based system may not be described mathematically, one should pay special attention to the stability issue. Therefore, the overall system stability is proved. Solution existence and error bounds are also analyzed. The good performance of the knowledge–based controller is verified via simulations for both linear and nonlinear systems.

The design of a general–purpose predictor is discussed in Chapter 4. By "general–purpose", we mean that a predictor is suitable for MIMO (multiple–input multiple–

output) systems with linear, nonlinear, time–invariant, and/or time–varying properties. Neural networks are used to design such a predictor. The following aspects are investigated: capability of neural networks for approximate prediction, basic structure of the neural network as a predictor, tracking a time–varying system, error analysis, and a training algorithm for an NN with vector inputs and outputs. The NN–based predictor is tested extensively via simulation for the MIMO systems mentioned above.

Chapter 5 presents the design procedures of the knowledge–based coordinator (KBC). The KBC combines the results discussed in Chapters 3 and 4, and forms a high–level coordinator in a hierarchical structure. The basic idea is to estimate the effects of the coordination commands to subsystems using a predictor and to modify these commands through a search in a knowledge base in order to achieve the desired performance. The problem of multiple–system coordination is stated formally, and some basic principles of multiple–system coordination are reviewed. The proposed scheme and the assumptions are described in detail. To show how to apply the proposed KBC to actual problems, the coordinated control for both tightly–coupled and loosely–coupled multiple systems is studied. In this chapter, the coordinated control of two 2–link robots holding a single object is presented as an example of the tightly–coupled multiple–system. The purpose is to reduce the internal force exerted on the object by modifying the reference input of each robot using the KBC. Adding the KBC does not impose any constraints on the design of the robots' servo controllers. In other words, the robots' dynamic properties are figured in the coordination without affecting the internal structure and parameters of each robot's control system.

For loosely–coupled systems, the coordinated control of multiple robots working

in a common workspace to avoid collision is analyzed in Chapter 6. It is assumed that both the desired path and trajectory of each robot are specified by teaching separately and without considering collision avoidance. A robot is designated as the master or a slave. The master will follow its desired trajectory, and the desired trajectories and/or paths of the slaves will be modified by the KBC to avoid collision. Both cylindrical robots and revolute robots are considered.

The second scheme for multiple–system coordination is presented in Chapter 7 in which neural networks are used for direct control and coordination. One of the key problems in designing such a controller (coordinator) is to develop an efficient training algorithm. NNs are usually trained by using the output errors of the network, instead of using the output errors of the controlled plant. However, when an NN is used to control a plant directly, the output errors of the network are unknown, since the desired control actions are unknown. This implies that the conventional back propagation training algorithm cannot be applied to control problems directly. In this chapter, a simple training algorithm is developed for a class of nonlinear systems, and this enables the NN to be trained by the output errors of the controlled plant. Both an NN–based controller and an NN–based coordinator have been designed and tested via simulation. The NN–based controller was tested on a thermo–process and shown to perform well in the presence of long system time delay, nonlinearity of dead zone and saturation, and process noise. The NN–based coordinator was tested on the same example for the KBC — two 2–link robots holding an object — and achieved even better results than that in Chapter 5.

Chapter 8 investigates the issues of performance evaluation for real–time control systems. We analyze the effects of computing time delay on the performance of both general real–time digital control systems and the proposed coordinators. For

a given fixed sampling interval, the effects of computing time delay are classified into *delay problem* and *loss problem.* The performance measures in the presence of computing time delay are reviewed, and then the effects of the both delay and loss problems are analyzed in detail. Some common misunderstandings of the effects of computing time delay are also clarified. For the qualitative analysis of the computing time delay, a generic criterion is derived in terms of system stability. For quantitative analysis, upper bounds of computing time delay on system stability and performance are derived for a robot control system. These upper bounds can be used as extra constraints on controller design and selection of CPUs to implement the control algorithm.

Finally, Chapter 9 summarizes the main contributions of this research and concludes the dissertation.

CHAPTER II

LITERATURE SURVEY

This research covers three subjects: multiple–system coordination, intelligent control, and neural networks.

2.1 Survey of Multiple–System Coordination

Early work on the basic theory of multiple–system coordination was summarized in [LMO82] and focused on large scale systems. Some conceptual principles were proposed including hierarchical system structures, goal coordination, model coordination, and interaction balance principle. Dynamic programming, mathematical programming, and other methods were also discussed as mathematical tools used for solving the coordination problem. However, most of the work in [LMO82] addressed only conceptual interpretation, but little has been said on applications. [Ozg89] surveyed and briefly analyzed existing approaches to decentralized and distributed control design for large scale systems including system and controller structures, intelligent control approaches, and implementation problems.

In recent years, research on coordinated control has focused on some specific applications, such as multiple–robot coordination. Though most designs of multiple–robot coordination are not in a hierarchical form, recent research in "multiple robots

holding a single object" is briefly reviewed below as an example of application–oriented work.

2.1.1 Multiple–Robot Coordination

When two or more robotic manipulators hold a single object, many problems occur due mainly to redundancy. If the object is assumed to be rigid and there is no relative motion between the end–effectors and the object, then they form a closed chain. In this closed chain, each end–effector with six DOFs (degrees of freedom) can exert six components of forces and moments on the object. However, the resulting motion of the object has only six components. Therefore, we have to control 12 variables (for the case of two manipulators) to get a result of six variables. In other words, due to kinematic constraints of the closed chain, some DOFs are lost. The number of these lost DOFs equals the number of DOFs gained to control internal forces of the closed chain [KL88], that is, the force/torque exerted on these DOFs will not cause any motion of the object, but will generate internal forces.[1] Detailed dynamic equations of the closed chain are given in [KL88], [Nak88], [ZL88], and [TBY88]. These internal forces must be controlled. Since load distribution to each joint is not unique due to redundancy, the joint torque needed to eliminate the internal forces are not unique either. Moreover, two manipulators may have different maximum loading capability due to different configurations, implying that the best load sharing may not necessarily be an equal share. The purpose of coordination is to control the internal forces while distributing the load and tracking a desired trajectory.

The load sharing problem can be stated as a constrained optimization problem.

[1] Henceforth, the term *"force"* will be used to mean force and torque, and the *"position"* means position and orientation, except as noted elsewhere.

[KL88] tried to separate the DOFs that contributed to the motion of object and that caused internal forces. Then master/slave scheme can be interpreted as a special case where the master controls the motion of the object while the slave controls internal forces exerted on the object. Load distribution was achieved by minimizing a quadratic function of joint torque and forces exerted on the object. The work in [LZ88] and [ZL88] dealt mainly with the load distribution problem for a master/slave scheme. They derived the optimization problem from the dynamic equation of each robot and the object obtained $2n+6$ linear scalar equations with $2(n+6)$ unknowns for joint torque and forces exerted on the object, where n is the number of DOFs of each robot. Associating these equations with constraints such as joint torque limits, the problem can be solved by some known optimization algorithms, like nonlinear programming. Hsu presented a hierarchical structure dealing with part–matching tasks with multiple robots [Hsu89]. The load sharing problem was solved by minimizing the weighted norm of the force applied to the object. The difficulty of these optimization methods lies in their real–time implementation.

Because two manipulators must grasp the object firmly, a small position error may cause a large force error. System design should, therefore, emphasize the coordination of interaction among the manipulators and the object. Zheng *et al.* analyzed the constrained relation between two manipulators, that is, given the position, velocity, acceleration and joint torque of the leader, how can those of the follower be solved [ZL85]. The generalized joint torque of the leader and follower can be computed, and used for coordinated control of the two manipulators. A nonlinear feedback control scheme was proposed in [TBY86], [TBY87] and [TBY88]. By applying nonlinear feedback, the dynamic equations of a two–robot system was linearized and decoupled. Yun discussed the case where two manipulators hold a single object and move it

along a surface while exerting a certain amount of contact force [Yun89]. Here nonlinear feedback was designed to linearize and decouple the nonlinear system, and a controller was designed based on the linearized system.

Adaptive control is also developed for coordination, for example [Ser88], [Koi85], [WKD89] and [MB89]. In [Ser88], a task–related Cartesian frame is set for both manipulators, so that the desired trajectory and force for each manipulator may be expressed relative to this common frame and coordination is achieved when the desired trajectory is generated. Then each manipulator acts as though it were carrying out commands alone in the Cartesian frame. In other words, the controllers work independently and the coupling effects between the two manipulators through the load are treated as disturbances which are then rejected by the adaptive property of a linear adaptive controller [Ser87]. Finally, the virtual forces in Cartesian space are transformed into joint space by a Jacobian matrix. A scheme using self–tuning control was proposed in [Koi85] in which each manipulator was represented by an ARMAX model. The system output vector included the position, orientation, linear and angular velocity of the end–effector, and the control inputs were joint torque. The self–tuning algorithm was used to minimize a certain cost function.

To eliminate the interactive force exerted on the object, [Pit88] designed a load–sharing controller. It is equivalent to adding two force feedback inputs to the computed torque algorithm so as to compensate for the force caused by the load and interaction between two manipulators. This algorithm was originally developed for motion control of a single robot with an unknown load. However, when it is applied to the coordinated control of two manipulators, note that not only the load is changed but also that the motion is constrained. This mode is different from that of one manipulator. Thus, it may not perform as well as in the case of a single ma-

nipulator. [Hay86], [Uch87] and [UD88] extended the hybrid position/force control algorithm, which is well–known for single robot compliant control, to multiple–robot coordination. In [KY88], the master/slave and hybrid position/force schemes were compared, and they concluded that the hybrid control is preferable. [AS88] extended the work of kinematic resolved–rate control to multiple–robot coordination. A force feedback loop was closed around the kinematic controller for each manipulator. Each robot was force–monitored to prevent it from imposing excessive stress on the object. Load distribution was handled by minimizing a quadratic cost function concerning the forces in task space.

The work in [KT88] is based on an unstructured model of $\boldsymbol{y}_i = \mathbf{G}_i(\boldsymbol{e}_i) + \mathbf{S}_i(\boldsymbol{f}_i)$, where $\boldsymbol{y}_i$ is the end–effector position in a global coordinate frame, $\boldsymbol{e}_i$ the input trajectory vector and $\boldsymbol{f}_i$ the external force measured in the same frame. $\mathbf{G}_i$ and $\mathbf{S}_i$ are L_p stable, linear mappings but their internal structures are not specified. Controllers are designed such that one controls the position and the other controls the force.

2.1.2 Comments

Recently, the National Science Foundation sponsored a Workshop on Coordination of Multiple Robot Manipulators: Planning, Control, and Applications [KB88]. One of the recommendations for future research is dynamical modeling and control systems in CMM (coordinated multiple manipulators) for which some important topics are modeling and simultaneous force/position control, parallel algorithms for control of CMM, and adaptive control of CMM.

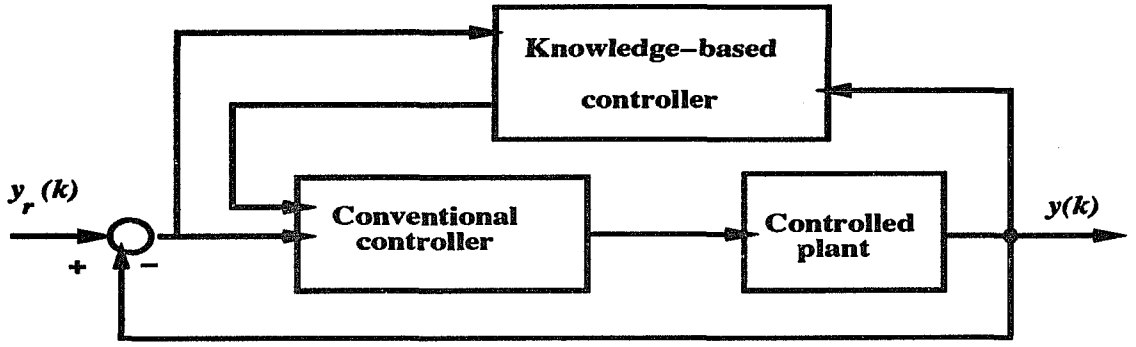
Many coordinated control schemes have been proposed. However, it was found that almost all of them attempted to solve the problem by improving the performance

of the servo control systems. This may not always work well because of the tracking error, especially for an imperfect trajectory generated by teaching. Currently, most commercially–designed robots are position–controlled or position–force controlled, and usually adopt PID (proportional, integration and derivative) controllers. This kind of control systems may not be suitable for those applications which require cooperation of multiple robots. Is it possible to coordinate two industrial robots with only a high–level coordinator without affecting the internal structure and parameters of low–level controllers by adding the coordinator? Answering this question is the major part of the proposed research.

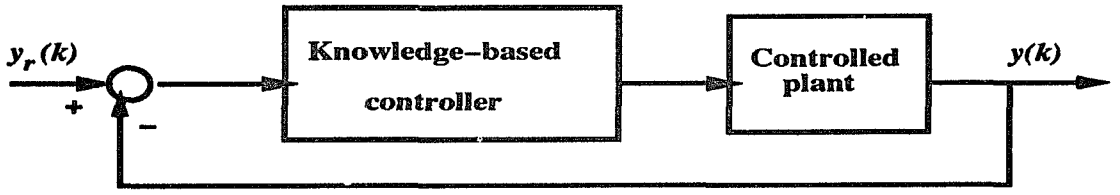
2.2 Survey of Intelligent Control

Since the first IEEE International Symposium on Intelligent Control (IC) in 1985, numerous papers have been published in this field. For a history of IC and its classification criteria, see survey papers [JS88], [Eld88], [APW88], [Mey87], [Sar88], and [GP89]. In what follows, only recent relevant works are briefly reviewed.

The major tools used for IC are the techniques developed in AI (artificial intelligence) — especially expert systems (rule–based systems, knowledge–based systems), and fuzzy set theory. Generally, the knowledge representation and reasoning methods established in AI are useful for making high–level decisions. This seems to be the reason why most control applications of AI have emphasized the development of computer–aided control system design packages, supervisory process operations, production planning, and so on. At the servo control level, an expert system seems unsuitable because of its difficulty in real–time computation. However, since knowledge from human experts can be easily represented by a set of production rules, application of IC in the servo control level has established two major branches. An expert



(a). Parameter–adaptive intelligent controller.



(b). Performance–adaptive intelligent controller.

Figure 2.1: Basic structures of intelligent control.

system can be combined with a conventional controller and used for controller tuning, fault diagnosis, and control system restructuring. This scheme intends to extend the range of conventional control algorithms by encoding general control knowledge and heuristic searches. Such schemes will henceforth be referred to as *parameter–adaptive IC* or *expert control* and are shown in Fig. 2.1 (a). The other schemes attempt to simulate human cognitive ability, which may require deep knowledge acquired from the plant dynamics and/or operation experiences. A set of production rules may be used to represent a qualitative model of a plant and to associate control operations with different situations. In this case, fuzzy set theory is often used for information extraction on which qualitative reasoning is based. This scheme is presented in Fig. 2.1 (b) and will henceforth be called *performance–adaptive IC* or *fuzzy control* or *qualitative control.*

2.2.1 Parameter–Adaptive Intelligent Control

[ABL88], [KBVB88] and [Lit90] presented typical structures of parameter–adaptive IC. In [ABL88], an IC is designed to tune a PID controller. Six different features are monitored: overshoot, rise time, settling time, peak height ratio, average value of the first two local minima and maxima, and the number of local maxima detected before settling down. Two sets of production rules are established on the measurement of these features. One is called alarm rules which will be fired when instability is identified. The other rules will be fired to re–tune the PID parameters. An algorithm, called the *membership–based tuning algorithm*, is used to quantify this tuning process. Another expert tuner of the PI controller is presented in [PJM87]. Its main objectives are wind–up protection of the I part and tuning of the PI gains. The transient response of a closed–loop system is characterized into nine categories including too–low–monotone, too–low–oscillatory, and so on. Moreover, the open–loop response is described by eight categories and the nonlinearity with eight categories. Similar to [ABL88], stability is monitored by the knowledge base.

Under the category of parameter–adaptive IC, an expert system can also be used to form a supervisory control system. [NKR89] designed a controller bank which consisted of a multiple–model adaptive controller and a model reference adaptive controller. An expert system is used to make a decision on which controller should be used. In [MW90], a rule–based system is developed to compensate for the dead–zone nonlinearity in a position–control system. As a supervisor, the rule–based system chooses control algorithms from the controller bank according to operating conditions. Employing the concept of blackboard, [Whi89] proposed the framework of a knowledge–based system with hierarchical structure. The global blackboard and two control modules form the top level and direct the operation of whole system. In

the second level, various tasks are executed, including fault diagnosis and controller tuning. The bottom level is implemented with a programmable logic controller which controls the process.

2.2.2 Performance–Adaptive Intelligent Control

Performance–adaptive IC is designed to replace the conventional controller. With qualitative reasoning in the controller, it must be integrated into the plant containing quantitative measurement and control signals. A typical design is to apply fuzzy set theory. First, the measurement of system output is fuzzified through a membership function so that qualitative reasoning can be conducted in the controller. Then the output of the controller is defuzzified into the actual control signal of an actuator. Knowledge is represented by a set of production rules. However, a fuzzy controller is usually implemented by look–up tables, to achieve high computational efficiency and to avoid unnecessary inference processes. Summaries of fuzzy control schemes are presented in [GP89] and [Lee90]. With a brief overview of the theory of fuzzy logic and rule–based control, [Ber88] described the major differences between the construction of fuzzy, rule–based controllers and conventional designs. [LL89] compared the performance of a fuzzy controller for servomotors with both the PID controller and the MRAC (model reference adaptive control) in terms of steady–state error, settling time and response time. Except for the basic structure of fuzzy control mentioned above, alternative designs have also been proposed. For example, [KV89] introduced a reference model into the design of a fuzzy logic controller for a linear system. [AW90] discussed the basic concepts of qualitative modeling and reasoning in process control. It was pointed out that a controller using a pure qualitative model of a process is inefficient in practice. To improve the performance of control

systems, the qualitative model should be combined with process–related qualitative knowledge.

Under the category of performance–adaptive IC, the principle of learning control can also be implemented [GJ88]. In control theory, a learning controller is a trial–and–error mechanism that repeats a fixed procedure. [GJ88] replaced this fixed procedure by a knowledge–based controller such that the intermediate output errors are treated as knowledge. The error and error increment are quantified into eleven intervals, such as negative–big, positive–low–middle, and so on. Based on this, an 11×11 table is formed corresponding to 121 rules. For control problems, control actions can be learned from the process directly or designed according to a model which is learned from the process [KKG88], [GW88]. In [KKG88], three expert systems form the controller: the goal selector, the identifier, and the adapter. Each learned model is valid within a subspace of parameters. The control rules are learned, allowing the system to recognize the qualitative region to which the controlled plant belongs at each time.

There are two important issues in designing fuzzy controllers: selection of a membership function and design of a knowledge base. Both of them depend on the knowledge extracted from experienced operators, on an understanding of the physical process of the plant, on the experimental results, and so on. [IS90] presented a scheme for the optimal design of membership functions. [Bat89] designed a supervisor to change the membership functions based on a certain performance index. Therefore, different operation situations can be controlled with a self–tuning property by switching the membership functions. Moreover, considering the fuzzy controller as a nonlinear time–varying controller and the plant as a linear time–invariant system, the stability of the closed–loop system was analyzed by using the circle criterion of

nonlinear control theory. Production rules can be derived from an analytical model. However, due to the approximation, simplicity and/or uncertainty of the model, the resulting rules may not achieve the desired performance. On the other hand, the rules from a human expert may lack the completeness to guarantee the optimality. Keeping this problem in mind, [Isi87] derived rules from experiments and implemented minimum time control in a mobile robot. In [LK87], two kinds of rules are designed: *training rules* and *machine rules.* The training rules are represented by a set of production rules which directly map the position and velocity errors of the controlled plant to the linear motion of the control mechanism. The machine rules are learned from the accumulated experience of control. The key problem is how to generate the "IF" part of the rules and keep consistency with existing rules.

Another important problem in designing an IC is how to characterize system performance. As discussed above, system output (or state) error and error increment, and the quality of step response are commonly used to express system performance. Besides these, [JD87] suggested to use the estimated, dominant pole locations of a closed–loop system as the characteristics of system performance, even though no knowledge base was built on it. [KBVB88] arranged output error and its derivative in a phase plane, and the goal was to control the system reaching the origin of this plane. The phase plane was divided into 48 areas and based on which the control rules were designed.

2.2.3 Comments

For a successful design of IC, the following features are important: (1) What are the characteristics used to express the performance of a system? (2) How is qualitative knowledge extracted from quantitative sensor data? and how is the result

of qualitative reasoning quantified into the quantitative control signal of actuators? (3) How is the knowledge base designed? (4) How are the rules of learning from experiences implemented? (5) How is system stability analyzed and guaranteed? From the above review, we note that only few papers analyzed system stability. The difficulty in applying conventional stability theory may come from the qualitative description of the plant model and the qualitative reasoning about the control actions.

It is necessary to compare the schemes of intelligent control with conventional control designs. [BCLM89] presented such a comparison, that is, fuzzy control was compared with state feedback control via a cart–pole balancing problem. For a given application, deciding which type of controllers should be used may depend on the following criteria: design complexity, completeness, robustness, performance, and modification of the controller. Berenji pointed out that fuzzy controllers may need significant calibration efforts to adjust membership functions and modification of the knowledge base may also require considerable effort. Another problem is how to integrate an IC with existent conventional control systems. At present, numerous IC designs have been proposed from servo control to the top level supervision. However, actual on–line implementations are scarce. One of the reasons is the lack of integration between IC and conventional control systems [Arz89]. [Arz90] discussed different schemes for this purpose based on the application of G2 — a real–time expert system tool.

It is suggested that IC could be used to complement the conventional designs of process control, especially either as a backup to conventional controllers or as a means of improving man–machine interface. Comparing with the controllers based on fuzzy logic and rule base, controllers designed using neural networks are a new type of IC and will be reviewed in the next section.

2.3 Survey of Neural Networks

The potential of NNs for control applications lies in the following properties: (1) they could be used to approximate any continuous mapping, (2) they can reach this approximation through learning, and (3) parallel processing and fault tolerance are easily achieved.

2.3.1 General Theory of Neural Networks for Control Applications

There are several survey papers which thoroughly reviewed the history and various applications of NNs. [Lip87] is such an earlier paper in which six important NN structures are discussed: the Hopfield net, the Hamming net, the Carpenter/Grossberg net, the single layer perceptron, the multilayer perceptron, and Kohonen's self organizing feature maps. The first three are suitable for binary inputs while the others are for continuous–valued inputs. They are also distinguished by training with or without supervision. Recently, [Cam90] and [Mel89] presented more detailed description of the major NN structures and algorithms, and provide a good starting point to study neural networks. [Koh87] reviewed the history of neural modeling and pointed out that it is natural to use NNs for all pattern recognition problems and sensor–motor control problems. Moreover, a general theory and methodology on the training of NNs is presented in [WM89]. [Fra89] gave a chronological review of the development of NNs in control applications and provided comparisons with adaptive control techniques. [KH89] surveyed the application of NNs in robotics, and concluded that NNs are suitable for task planning, path planning, and path control.

The most popular NN structures for applications are multilayer perceptron with the BP (back propagation) algorithm and the Hopfield net. Both are trained with supervision, and multilayer perceptron seems more suitable for control applications

because continuous–valued input could be applied. [Wer88] summarized a generalized formulation of BP and discussed network architectures and applications. A detailed version of the BP algorithm is presented in [RM86]. BP is now successfully used for pattern classification, though the original development of BP placed more stress on control [Wer89a]. The importance and application of NNs to control and system identification are addressed in [Wer89a] and [Wer89b], and five dominant architectures now in use for control are also discussed: supervisory control, inverse dynamics control, control with BP through time, and two designs based on adaptive critics and reinforcement learning.

Applying NNs to process control has recently become widespread. Most applications adopt the multilayer perceptron with the BP algorithm, due mainly to the property that a multilayer perceptron could be used as a universal approximation of continuous functions. Moreover, its well–established network architecture and simple training algorithm enforce its potential for control applications. [HN89] presented a survey of the basic theory of BP covering architecture design, performance measurement, function approximation capability, and learning. It is proved that any L_2 function from $[0,1]^n$ to $\mathbf{R}^m$ can be implemented to any desired degree of accuracy with a three–layer[2] perceptron. [Cyb89] and [Bar89] proved and summarized some approximation properties of NNs. It was shown that for any continuous function, f^*, on a compact subspace of d-dimensional Euclidean space, there exists a sequence of network functions, f_n, that converges uniformly to f^*. In general, they concluded that perceptron with one hidden layer and an arbitrary continuous sigmoidal function can approximate continuous functions with arbitrary precision, if there are no constraints on the number of nodes or the size of the weights. [SW89] reached the

[2]Within this dissertation, the term "n-layer" means that the network has $n-2$ hidden layers.

same conclusion and proved that the activation function in hidden nodes is not necessarily a sigmoid but a general nonlinear function. Moreover, [Son90] compared the representational capabilities of three–layer and four–layer perceptron and concluded that nonlinear control systems can be stabilized using four layers but not, in general, using three layers.

All of the work mentioned above dealt with the existence of NNs for universal approximation. Usually, the numbers of nodes in input and output layers are determined according to the input/output patterns. Therefore, an important problem is how to determine the number of hidden nodes required to approximate a given function with desired accuracy. For a three–layer network, [Ara89] shown that if the hidden nodes use binary values, $J-1$ hidden nodes are the necessary and sufficient condition to obtain an arbitrary mapping for given J input patterns. Arai also concluded that to get an arbitrary mapping for continuous–valued inputs with finite hidden nodes, more than three–layer networks must be taken into account. [GWG89] proposed a method of estimating the number of hidden nodes required by any three–layer perceptron performing binary mapping. [KH88] provided an analytical solution to the problem of choosing the number of hidden nodes and the best learning gains.

2.3.2 Applications of Neural Networks to System Control

Most earlier papers on NN applications to system control are summarized in the survey papers mentioned above. In what follows, only recent developments are reviewed. Generally, the control system architectures using NNs can also be classified into two groups: *performance–adaptive* control and *parameter–adaptive* control. In each group different networks and training algorithms could be adopted. Most pa-

pers appear to have preferred performance–adaptive control to parameter–adaptive control.

There is a type of NN controllers with a similar structure to MRAC, but the reference model is used only to train an NN. In [GS88], a four–layer perceptron was used as the controller, and the reference model was a human teacher. In [BKG89], an unsupervised approach to nonlinear system control was proposed, and the input signals of the NN were the output of a reference model, and the desired and actual output of the system. It was shown that stability was maintained under the same bounds as the guaranteed stability of linear controllers.

Another scheme in performance–adaptive control is training an NN to learn the inverse of a system. Certainly, this requires that the system is invertible. [CP89], [PSY88], [Els88], [ZPK89], [BB89], and [LGI89] are such examples. However, the weights of the network need to be updated using the network's output error which is unknown when the NN is serially connected to a controlled plant. This implies that the BP cannot be applied directly. In [CP89], the controlled system was treated as an additional, unmodifiable layer, and the output error of the network was computed from the system–output error. In [PSY88], the system–output error was propagated back through the plant using partial derivatives of the plant at an operating point. Elsley [Els88] used a three–layer perceptron to learn the inverse Jacobian of a system, letting each input activate four nodes at the NN's INPUT layer, and each was sensitive to some range of input values. Then the outputs of the four nodes at NN's OUTPUT layer were combined to form the control signal. The network was trained with a correct inverse Jacobian. However, in practice, even if the system is invertible, the inverse control scheme may be not acceptable. For example, if the system is in non–minimum phase, then the resulting design is not internally stable. In [YG89],

a context–sensitive network was designed to learn the inverse Jacobian matrix of a PUMA 560 robot. The robot's end–effector and joint velocities are related by $\dot{\boldsymbol{X}} = \mathbf{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$, where $\mathbf{J}(\boldsymbol{q})$ is the Jacobian matrix, $\boldsymbol{q}$ and $\dot{\boldsymbol{X}}$ are the inputs of an NN, and $\dot{\boldsymbol{q}}$ is its output. For a six–DOF robot, there are 12 inputs and six outputs. The basic idea was to partition the inputs into two sets for two networks. $\boldsymbol{q}$ and $\dot{\boldsymbol{X}}$ were used as the inputs of a *context network* and a *function network*, respectively. Then, the outputs of the context network were used to set up the weights of the function network. In other words, the function network was programmed by the context network. The training algorithm was the extension of the BP. The invertibility of nonlinear systems was discussed in [Gu90], and a sufficient–input criterion for designing an NN to learn a system's inverse was established.

Asada used NNs for robot compliant motion control, where the compliance was treated as a nonlinear mapping from a measured force to a motion correction [Asa90]. A three–layer perceptron was used to learn the mapping. As an example, the peg–in–hole problem was presented. For this example, it is critical to recognize which edge(s) of the hole the peg contacts. The NN must be able to detect and discriminate individual contact. Since the original structure of multilayer perceptron is suitable for a static mapping, two modified schemes were suggested in [YY90] using a recurrent feedback loop with a time delay function. Moreover, the stability of a system with NN controllers learning system inverse were analyzed using the Lyapunov method. Miller proposed a learning control technique which is an extension of CMAC (cerebellar model articulation controller), and the controller was tested on a five–link robot [MHGK90]. Kraft *et al.* [KC89] compared an NN controller with an MRAC and a self–tuning regulator. The NN controller had a similar structure to CMAC. They concluded that NNs can solve some problems for which traditional

adaptive controllers may not perform well.

An NN can also be made to learn some system properties, based on which a controller is then designed. In [BM89], the output prediction of a system was computed by a three–layer perceptron. Then, using the prediction, a controller was designed to minimize a cost function. In [HA89], NNs were also used for model predictive control, in which the NN was trained to simulate the dynamics of a plant. Fadali *et al.* presented a scheme of bang–bang control for robotic manipulators, in which a multilayer perceptron was used to learn the switching time [FAET90]. In [Pou89], two types of NNs were proposed for control: forward NNs and recurrent multilayer NNs. Training rules were designed using the same methods as those for nonlinear adaptive control systems, such as Lyapunov's stability theory. [NP90] used a multilayer perceptron with the BP algorithm for system identification. The NN was trained to attain the same dynamic behavior as the controlled plant. Then a controller was designed by using the NN's outputs to cancel the nonlinear part of the controlled plant and including the same terms of a reference model. [HF91] reported a method of feedback linearization of the controlled plant using NNs. The NN's role is to directly approximate the Lie derivatives which form the state feedback control. The NNs consist of two layers; the training algorithm is derived based on Lyapunov stability type argument.

A typical structure of parameter–adaptive control with an NN was presented in [GEK88]. An NN was used as an estimator to adjust the parameters of a servo controller. As an example, a two–element, Cohen–Grossberg type NN was trained to tune the parameters of a PD controller. In [Swi89], a PID controller was tuned by a multilayer perceptron. In [JL90], a multilayer perceptron was used to estimate the payload of a robot during high speed motion. The estimation problem was stated as

a pattern–recognition problem based on trajectory tracking errors, and an adaptive model–based controller was designed.

2.3.3 Comments

NNs can serve as either a controller or an observer. An NN may perform best as a controller. However, during the training period, the system may lose robustness due to the random initial weights of the NN. As an observer, the NN is used only for system identification, but the problem of initial weights is usually not serious. Most of the works cited above are in the form of indirect control, that is, the controller is designed based on the results of system identification or parameter estimation which is realized by NNs. On the other hand, when an NN is used to control a plant directly, more difficulties arise due to the lack of a well–developed algorithm. Generally, an NN is trained by minimizing its output error. However, in the case of direct control, the outputs of the NN are the control inputs of the plant. Therefore, the desired outputs of the NN are unknown, since the desired control actions are unknown. This implies that the output errors (training errors) of the NN is not available and the standard BP algorithm cannot be used directly. With this in mind, a simple scheme for direct control (coordination) for a class of nonlinear systems has been developed in Chapter 7 of this dissertation.

Applications of NNs in control are far from being complete, though some papers have been published and claimed success. Moreover, little work has been done to compare NN approaches with traditional control designs. Obviously, more theoretical analyses, case studies, and experiments are needed.

CHAPTER III

HIERARCHICAL KNOWLEDGE–BASED CONTROLLER FOR A SINGLE SYSTEM

3.1 Introduction

Conventional control theory is based on mathematical models that describe the dynamic behavior of controlled plants. These models usually consist of a set of linear or nonlinear differential/difference equations, most of which are derived under some forms of approximation and simplification. However, complexity, model uncertainty and/or parameter uncertainty of the controlled plants often make the controllers very complicated. On the other hand, human operators do not always handle a system control problem with a detailed mathematical model, but rather with a qualitative or symbolic description of the controlled plant. This fact calls for the need of IC (intelligent control) for complex systems. In Chapter 2, we know that most related IC designs can be referred to as parameter–adaptive or performance–adaptive IC. The fundamental difference between these two lies in the goal of its knowledge base. The typical structures of a parameter–adaptive IC and a performance–adaptive IC are sketched in Fig. 2.1 (a) and (b), respectively. Though many IC schemes have been proposed, it is difficult to compare them because of lack of theory and lack of a universal performance criterion. As we pointed out in Chapter 2, for a successful

design of IC, the following issues are important:

1. What are the characteristics used to express the performance of a system?
2. How is qualitative knowledge extracted from quantitative sensor data? and how is the result of qualitative reasoning quantified into the quantitative control signal of actuators?
3. How is the knowledge base designed?
4. How are the rules of learning from experience implemented?
5. How is system stability analyzed and guaranteed?

In contrast to the parameter–adaptive and performance–adaptive ICs, we propose a new hierarchical knowledge–based controller in this chapter. The basic principles of this controller will be described in Section 3.2. The characteristics of the low–level subsystem and the modification of reference input are also presented in this section. Section 3.3 is a detailed description of the knowledge–based controller, which includes knowledge representation, existence of the solution, and inference process. The stability of the knowledge–based controller is analyzed in Section 3.4. In Section 3.5, the procedure for designing a predictor is discussed briefly, and a detailed error analysis is introduced which gives the lower and upper bounds of the trajectory tracking error. The simulation results of the knowledge–based controller is also shown in that section with promising performance. This chapter concludes with Section 3.6.

3.2 Design Principles and Characteristics

Basic Principles

A control system is evaluated by examining its response to typical, pre–planned

trajectories, such as step, slope, parabola and/or sinusoidal signals. There are two ways to improve the performance of the control system. One is to set the desired trajectory as the system reference input and to redesign the internal structure of the servo controller in order to track the reference input precisely. For a complex control system, if this approach is used, the servo control level will become more complicated, and the fine tuning of the controller parameters will be extremely tedious (particularly for nonadaptive schemes). Moreover, there are some design trade–offs to consider, such as the one between rise time and maximum overshoot. The other way is to choose and adjust a reference input such that the controlled system tracks the pre–planned trajectory. This forms a hierarchical structure, but requires little change in the internal structure of the servo control level. This is exactly what a hierarchical system is supposed to be; each level in the hierarchy is independent and does not affect the internal structures of other levels.

In other words, in the high–level controller's view, the low–level subsystem is nothing but a mapping from the reference input to the system output. Therefore, there are two ways to improve the performance of the subsystem. One is to modify the map itself, that is, some parameters or even the structure of the servo controller. This requires the high–level to know the detailed internal structure of the lower level. The other way is to modify only the domain of the map (that is, the reference input of the lower level) without requiring any detailed knowledge of the subsystem's structure. Considering the generality and the inexactness of the structure of the low–level subsystem, we have adopted the latter approach. This also coincides with the principle of increasing intelligence while decreasing precise knowledge of the lower levels as we move up the levels of hierarchy.

It is assumed that the servo controller is designed independently of the high–

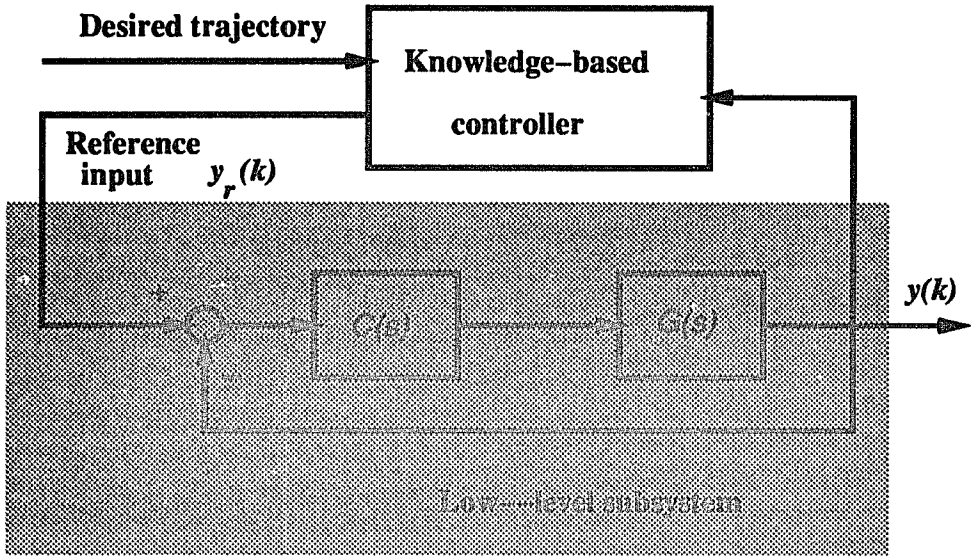


Figure 3.1: Structure of the hierarchical knowledge–based controller.

level controller, and that its dynamic structure and parameters are unknown to the high–level controller. Therefore, we shall design a knowledge–based controller (as a high–level controller) which modifies only the reference input to the subsystem as shown in Fig. 3.1, in which $G(s)$ and $C(s)$ are the controlled plant and a conventional controller, respectively. As a result, the internal structure and/or parameters of the (low–level) servo controller are not altered at all, thus imposing no constraints on the servo control level. This will, in turn, enable commercially–designed servo controllers to perform more sophisticated tasks than originally intended.

Characteristics of the Subsystem and the Modification Process

To design a knowledge–based controller, one has to specify the input space of the knowledge base, or choose a typical representation of the system's dynamic characteristics. The most commonly used components are output error and/or error increment, and the standard figures of step response. However, to express system characteristics more directly and to eliminate the undesirable effects of time delay, we propose to use predicted system outputs from which suitable reference inputs

are determined. It is assumed that the lower level of hierarchy is a well–designed, closed–loop control system, and the system output prediction is available.

Now, the problem is how to modify the reference input in order to make the system output track the desired trajectory. Using the system output prediction, the desired performance can be achieved by iterative trial as is done in learning control. Note that learning control is usually used for a repetitive trajectory and needs a learning period during which an unacceptable output error could occur. By contrast, this knowledge–based controller is designed for an arbitrary trajectory and has to complete the iterative learning process in each sampling interval. The modification process is to

1. give a reference input,
2. compute the predicted system output,
3. calculate the predicted tracking error by comparing the prediction and the desired trajectory, and
4. modify the reference input based on this error.

Note that actions taken in a control system are in general irrecoverable; that is, each reference input to the servo controller is the final decision at each sampling interval and cannot be undone. However, combining prediction and modification allows us to analyze the *anticipated* consequence of each reference input, thereby at least partially solving the irrecoverable problem.

This modification process can be formalized as follows. Let $y_r^i(k)$ be the reference input, $y_d(k)$ the desired trajectory, and $\hat{y}^i(k+d/k)$ the d-step ahead prediction of the system output at time k, where the superscript i denotes the i-th iteration. The

reference input $y_r^i(k)$ is modified by

$$\begin{aligned} y_r^{i+1}(k) &= y_r^i(k) + K_0^i(k)\, e^i(k+d), \\ \text{where } e^i(k+d) &= \hat{y}^i(k+d/k) - y_d(k+d), \\ y_r^0(k) &= y_d(k), \qquad i = 0,\ 1,\ \cdots. \end{aligned}$$

$K_0^i(k)$ is the learning gain at time k during i-th iteration, and $K_0^0(k) = 0$. Then we get

$$y_r^{i+1}(k) = y_d(k) + \sum_{j=0}^{i} K_0^j(k)\, e^j(k+d).$$

Accurate tracking will be achieved by the iterative operation and the prediction, and this iterative operation must be completed in each sampling interval. To make the above modification feasible, the following conditions need to be met: (1) the iterative operation converges fast, and (2) the output prediction of the system is computable. Condition (1) is usually met because the lower level is a well–designed control system, and $y_r^i(k)$ is near the optimal point. Condition (2) will be discussed in the next chapter.

The low–level subsystem is equipped with some well–designed servo controllers, and assumed to be linear. Then, following an argument similar to the one in [TY86], we can prove that a learning gain $K_0^i(k)$ exists such that $e^i(k+d) \longrightarrow 0$ as $i \longrightarrow \infty$. Though such a $K_0^i(k)$ exists, due to lack of knowledge of the low–level subsystem, it is not easy to calculate the gain accurately. Moreover, the parameters and/or model uncertainties are not even considered, thus necessitating design of a knowledge–based controller.

3.3 Description of the Knowledge–Based Controller

Knowledge Representation

Using the predictor, the subsystem performance is characterized by the predicted tracking error and the current reference input. Therefore, the space of the predicted tracking error forms the input space of the knowledge base. The goal of the knowledge–based controller is to implement the modification process discussed thus far. It is not difficult to express this process by a set of production rules. The possible actions that the knowledge–based controller can take include: increasing the reference input, decreasing the reference input, and keeping the reference input unchanged. The problem is how much to increase/decrease and how to determine the bounds of the reference input. Because this scheme is based on the modification of the reference input according to the resulting predicted output, the internal structure and parameters of the low–level subsystem are not affected. This property allows us to consider the predicted tracking error, but not its derivative, as the system characteristics so as to simplify designing of production rules. The basic modification process can be represented by a decision tree as shown in Fig. 3.2.

The ij-th node is represented by $([a_j^i,\ b_j^i],\ c_j^i)$, where c_j^i is the quantity added to the reference input,

$$y_r^{i+1}(k) = y_d(k) + c_j^i,$$

and $[a_j^i,\ b_j^i]$ is the interval to be searched, and $a_j^i < c_j^i < b_j^i$ for all $i,\ j$. By giving the reference input $y_r^i(k)$, at any node $([a_j^i,\ b_j^i],\ c_j^i)$, the interval $[a_j^i,\ b_j^i]$ will be split into two subintervals $[a_k^{i+1},\ b_k^{i+1}] \equiv [a_j^i,\ c_j^i]$ and $[a_{k+1}^{i+1},\ b_{k+1}^{i+1}] \equiv [c_j^i,\ b_j^i]$, which form the two successor nodes of $[a_j^i,\ b_j^i]$. During the i-th iteration and at ij-th node, let $e_j^i(k)$ denote the predicted tracking error resulting from $y_r^i(k)$:

$$e_j^i(k) \equiv e^i(k+d) = \hat{y}^i(k+d/k) - y_d(k+d).$$

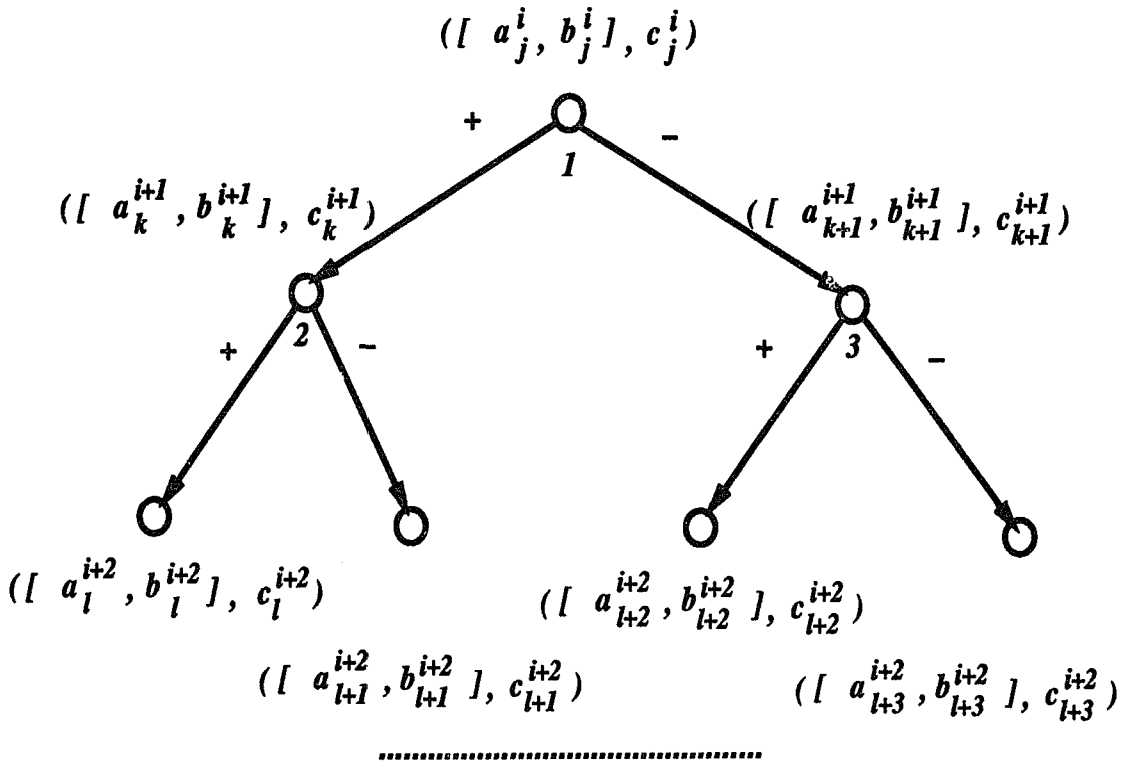


Figure 3.2: Decision tree

Then, c^i_j is computed as:

$$c^i_j = \begin{cases} b^i_j - (b^i_j - a^i_j)K, & \text{if} \quad e^i_j(k) < 0 \\ 0, & \text{if} \quad e^i_j(k) = 0 \\ a^i_j + (b^i_j - a^i_j)K, & \text{if} \quad e^i_j(k) > 0 \end{cases}$$

and $0 < K < 1$ is a weighting factor which determines the step size of the iterative operation. a^0_0 and b^0_0 are the pre–designed lower and upper bounds of the reference input modification, and usually $c^0_0 = 0$, that is, at the beginning, the reference input is not modified.

Solution Existence and Inference Process

The basic forms of the production rules are follows.

IF $e^i_j(k) < 0$ **AND** $|e^i_j(k)| > \epsilon$,

THEN increase c_j^i **AND** compute $y_r^{i+1}(k) = y_d(k) + c_j^i$.

IF $e_j^i(k) > 0$ **AND** $|e_j^i(k)| > \epsilon$,

THEN decrease c_j^i **AND** compute $y_r^{i+1}(k) = y_d(k) + c_j^i$.

IF $|e_j^i(k)| \leq \epsilon$,

THEN set $y_r^{i+1}(k) = y_r^i(k)$ **AND** stop the iterative operation.

$\epsilon > 0$ is a pre–specified error tolerance. Because the amount of modification to the reference input is bounded, or $a_0^0 < c_j^i < b_0^0$, for all i, j, there may be a case that $|e_j^i(k)| > \epsilon$ for all c_j^i. To avoid this situation, the desired trajectory needs to be carefully designed. For example, when the desired trajectory is a step function and the system time delay is equal to two sampling intervals, at $k = 0$ the continuous system response cannot have a jump no matter how large the reference input is. A reasonable choice of ϵ is another way to prevent this problem. This problem can be monitored by adding, for example, the following rule into the knowledge base:

IF $(|c_j^i - b_0^0| < \delta$ **OR** $|c_j^i - a_0^0| < \delta)$ **AND** $|e_j^i(k)| > \epsilon$

THEN change a_0^0 or b_0^0 automatically and continue the search, **OR**

ask the operator for an adjustment, **OR**

stop the iterative operation and choose c_j^i with

the smallest $e_j^i(k)$ as the best output.

Suppose the weighting factor K is set too small or too large, then the search for a proper c_j^i may take a very long time. This would not be acceptable if the required computation cannot be completed within one sampling interval. The case of the computation/search time exceeding one sampling interval is equivalent to having no solution. This case is monitored by:

IF the search time $> T_{max}$ **AND** $|e_j^i(k)| > \epsilon$

THEN stop the iterative operation **AND**

choose c_j^i with the smallest $e_j^i(k)$ as the best output, **AND**

modify the weighting coefficient K,

where T_{max} is a pre–selected maximum allowable search time.

Based on the structure of the decision tree, one can see that the simplest inference process is similar to forward chaining, starting from the root node. However, it may be learned after a period of operation that, for example, a positive augment c_j^i is always needed. If such a fact is learned, the inference process can start from any node with $c_j^i > 0$ and go forward or backward, depending on the sign of the predicted tracking error. Note that the backward search does not mean a reverse search, but rather intends to find a suitable node from which a forward search can begin. As soon as the forward search begins, the search process is not reversible.

3.4 Stability Proof

It may be easy to establish the stability of the low–level subsystem for a fixed reference input, because it is a well–designed closed–loop control system. However, this does not imply the stability of the whole system. See Fig. 3.3 for a block diagram of the knowledge–based control system. Both system poles and zeros are affected by the presence of the knowledge–based controller and the predictor. If the transfer functions for all the blocks in Fig. 3.3 are given, we may be able to derive the conditions for system stability. But this is not the case in reality: $C(s)$ and $G(s)$ may not be known accurately, and the iterative learning with the prediction and the knowledge–based controller do not form a simple feedback loop and cannot be expressed as simple mathematical transfer functions.

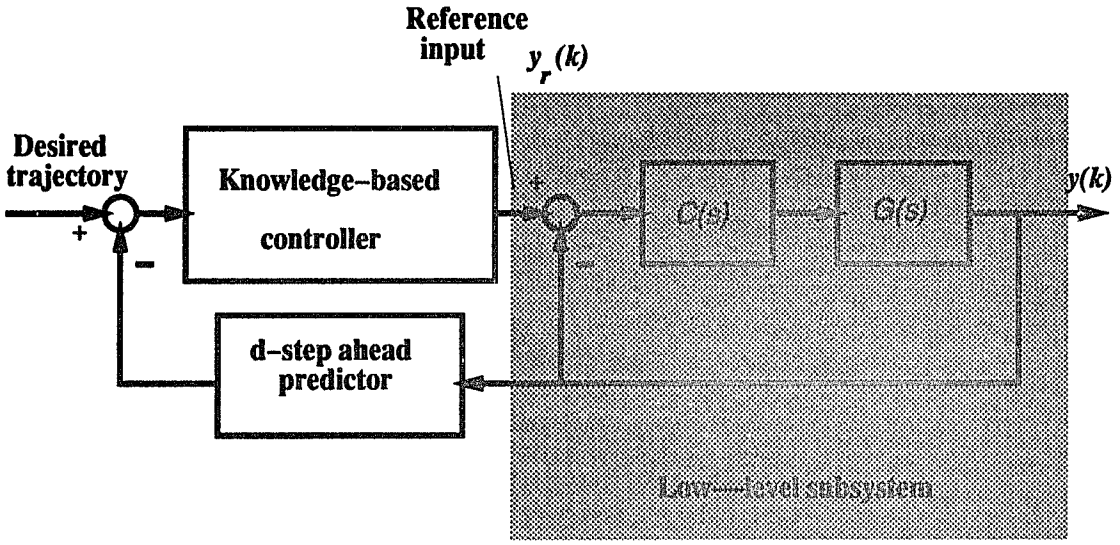


Figure 3.3: Block diagram of the knowledge–based control system.

If we suppose that the prediction gives the true system output, then let us consider the knowledge–based controller and the closed–loop subsystem. (The assumption of perfect output prediction is of course unrealistic and will be relaxed in our later discussions.) The knowledge–based controller can be viewed as a map $\mathbf{M}_0 : \mathbf{E} \rightarrow \mathbf{Y}_R$, specified by all the production rules, where $\mathbf{E} \subset \mathbf{R}$ is the space of predicted tracking error and $\mathbf{Y}_R \subset \mathbf{R}$ the reference input space. The low–level closed–loop subsystem is also a map, $\mathbf{L} : \mathbf{Y}_R \rightarrow \mathbf{E}$, which is specified by the desired dynamic properties of the servo controller. Because $\mathbf{L}$ represents a well–designed controller and a reference input, $y_r^i(k) \in \mathbf{Y}_R$, exists at time k such that the trajectory tracking error $e^i(k+d) = 0$, it is reasonable to assume that $\mathbf{L}$ is a linear map. The properties of the map $\mathbf{M} \equiv \mathbf{L}\ \mathbf{M}_0 : \mathbf{E} \rightarrow \mathbf{E}$ depends mainly on the properties of the map $\mathbf{M}_0$. In fact, all the antecedents of production rules are established based on the output prediction. If the predictor gives the true output, then the properties of the invariant map $\mathbf{M} : \mathbf{E} \rightarrow \mathbf{E}$ is determined solely by the knowledge base.

For system stability, all production rules in the knowledge base must form a

contraction map. More formally, we have the following theorem.

Theorem 3.1: Suppose (1) the output prediction of the low–level subsystem is computable and the predictor gives the true output, and (2) $\mathbf{L} : \mathbf{Y}_R \rightarrow \mathbf{E}$ of the low–level closed–loop subsystem is a linear map. If the map $\mathbf{M_0} : \mathbf{E} \rightarrow \mathbf{Y}_R$ is given by the decision tree, then the composite map $\mathbf{M} \equiv \mathbf{L}\ \mathbf{M_0} : \mathbf{E} \rightarrow \mathbf{E}$ is a contraction map.

Proof: The basic production rules can be represented in the form of

IF by the reference input $y_r^i(k)$, the predicted tracking error $e_j^i(k) < 0$

THEN increase c_j^i to get c_{k+1}^{i+1}, $c_{k+1}^{i+1} > c_j^i$,

IF by the reference input $y_r^i(k)$, the predicted tracking error $e_j^i(k) > 0$

THEN decrease c_j^i to get c_k^{i+1}, $c_k^{i+1} < c_j^i$,

where $e_j^i(k)$ is the predicted tracking error resulting from the augment c_j^i at the j-th node of the i-th level in the decision tree. These rules associate each predicted tracking error with a specified value of c_j^i. From a new augment c_k^{i+1} or c_{k+1}^{i+1}, the predicted tracking error $e_k^{i+1}(k)$ and $e_{k+1}^{i+1}(k)$ are then computed.

Since the decision tree is searched downward after finding a starting node in the tree, we want to show that this search process has the following property:

$$\mathbf{d}(c_l^{i+2},\ c_k^{i+1}) < \mathbf{d}(c_k^{i+1},\ c_j^i). \tag{3.1}$$

for all $i,\ j,\ k,\ l = 0,\ 1,\ 2, \cdots$, and $\mathbf{d}(\ ,\ )$ is a metric on $\mathbf{Y}_R$. Referring to Fig. 3.2, we get

$$\begin{aligned} c_k^{i+1} &= a_k^{i+1} + (b_k^{i+1} - a_k^{i+1})K = a_j^i + (c_j^i - a_j^i)K \\ c_{k+1}^{i+1} &= b_{k+1}^{i+1} - (b_{k+1}^{i+1} - a_{k+1}^{i+1})K = b_j^i - (b_j^i - c_j^i)K \\ c_l^{i+2} &= a_l^{i+2} + (b_l^{i+2} - a_l^{i+2})K = a_j^i + (c_k^{i+1} - a_j^i)K \end{aligned}$$

$$
\begin{aligned}
c_{l+1}^{i+2} &= b_{l+1}^{i+2} - (b_{l+1}^{i+2} - a_{l+1}^{i+2})K = c_j^i - (c_j^i - c_k^{i+1})K \\
c_{l+2}^{i+2} &= a_{l+2}^{i+2} + (b_{l+2}^{i+2} - a_{l+2}^{i+2})K = c_j^i + (c_{k+1}^{i+1} - c_j^i)K \\
c_{l+3}^{i+2} &= b_{l+3}^{i+2} - (b_{l+3}^{i+2} - a_{l+3}^{i+2})K = b_j^i - (b_j^i - c_{k+1}^{i+1})K.
\end{aligned}
$$

The metric defined on $\mathbf{Y}_R$ is given by

$$
\begin{aligned}
\mathbf{d}(c_k^{i+1},\ c_j^i) &\equiv |c_k^{i+1} - c_j^i| = |(a_j^i - c_j^i)(1-K)| \\
\mathbf{d}(c_{k+1}^{i+1},\ c_j^i) &\equiv |c_{k+1}^{i+1} - c_j^i| = |(b_j^i - c_j^i)(1-K)| \\
\mathbf{d}(c_l^{i+2},\ c_k^{i+1}) &\equiv |c_l^{i+2} - c_k^{i+1}| = |(a_j^i - c_j^i)(1-K)K| \qquad (3.2) \\
\mathbf{d}(c_{l+1}^{i+2},\ c_k^{i+1}) &\equiv |c_{l+1}^{i+2} - c_k^{i+1}| = |(c_j^i - a_j^i)(1-K)^2| \qquad (3.3) \\
\mathbf{d}(c_{l+2}^{i+2},\ c_{k+1}^{i+1}) &\equiv |c_{l+2}^{i+2} - c_{k+1}^{i+1}| = |(c_j^i - b_j^i)(1-K)^2| \qquad (3.4) \\
\mathbf{d}(c_{l+3}^{i+2},\ c_{k+1}^{i+1}) &\equiv |c_{l+3}^{i+2} - c_{k+1}^{i+1}| = |(b_j^i - c_j^i)(1-K)K|. \qquad (3.5)
\end{aligned}
$$

Suppose $0.5 < K < 1$, then at node 2, from Eqs. (3.2) and (3.3) we get

$$\mathbf{d}(c_l^{i+2},\ c_k^{i+1}) > \mathbf{d}(c_{l+1}^{i+2},\ c_k^{i+1}).$$

Taking the larger of $\mathbf{d}(c_l^{i+2},\ c_k^{i+1})$ and $\mathbf{d}(c_{l+1}^{i+2},\ c_k^{i+1})$ and comparing it with $\mathbf{d}(c_k^{i+1},\ c_j^i)$, we get

$$\mathbf{d}(c_l^{i+2},\ c_k^{i+1}) < \mathbf{d}(c_k^{i+1},\ c_j^i), \qquad (3.6)$$

because $(1-K)K < (1-K)$. Similarly, at node 3, from Eqs. (3.4) and (3.5) we get $\mathbf{d}(c_{l+3}^{i+2},\ c_{k+1}^{i+1}) > \mathbf{d}(c_{l+2}^{i+2},\ c_{k+1}^{i+1})$. Taking the larger of $\mathbf{d}(c_{l+3}^{i+2},\ c_{k+1}^{i+1})$ and $\mathbf{d}(c_{l+2}^{i+2},\ c_{k+1}^{i+1})$ and comparing it with $\mathbf{d}(c_{k+1}^{i+1},\ c_j^i)$, we get

$$\mathbf{d}(c_{l+3}^{i+2},\ c_{k+1}^{i+1}) < \mathbf{d}(c_{k+1}^{i+1},\ c_j^i), \qquad (3.7)$$

because $(1-K)K < (1-K)$. Both Eqs. (3.6) and (3.7) show that Eq. (3.1) holds.

When $0 < K \leq 0.5$, Eq. (3.1) can be proved similarly.

Because $\mathbf{L} : \mathbf{Y}_R \rightarrow \mathbf{E}$ is a linear map, the property of Eq. (3.1) in $\mathbf{Y}_R$ is preserved under the map $\mathbf{L}$ and has the form of

$$\mathbf{d}_1(e_l^{i+2},\ e_k^{i+1}) < \mathbf{d}_1(e_k^{i+1},\ e_j^i), \tag{3.8}$$

for all $i,\ j,\ k,\ l = 0,\ 1,\ 2, \cdots$ and $\mathbf{d}_1(\ ,\ )$ is a metric on $\mathbf{E}$. By Eq. (3.8), $\mathbf{M} : \mathbf{E} \rightarrow \mathbf{E}$ is a contraction map, and the theorem is proved. □

At each node in the structure of the decision tree, the rules always keep the search in the direction pointing to the node where the tracking error decreases. Because the iterative learning process is performed at each node, this is equivalent to the claim that the iterative learning process decreases the tracking error. As mentioned in Section 3.3, the inference process is not reversible, and thus, it is impossible to have an unstable system response.

3.5 Design of the Knowledge–Based Control System and Simulation

Design of a Predictor

As stated earlier, the low–level subsystem is equipped with a servo controller, and is assumed to have a linear response to the reference input. For such a linear system, there are several algorithms available for designing a predictor. For convenience, we have designed a self–tuning predictor in order to test the performance of the knowledge–based controller only. The principle of the self–tuning predictor is briefly stated below (see [KC81] or [CS88] for a detailed account). The low–level, closed–loop subsystem is represented by an ARMAX model:

$$\mathbf{A}(z^{-1})\ y(k) = \mathbf{B}(z^{-1})\ y_r(k - d_0) + \mathbf{C}(z^{-1})\ \xi(k) \tag{3.9}$$

$$\text{where}\quad \mathbf{A}(z^{-1}) = 1 + a_1\ z^{-1} + \cdots + a_n\ z^{-n},$$

$$\mathbf{B}(z^{-1}) = b_0 + b_1\, z^{-1} + \cdots + b_n\, z^{-n},$$

$$\mathbf{C}(z^{-1}) = 1 + c_1\, z^{-1} + \cdots + c_n\, z^{-n}.$$

d_0 is an index representing the time delay, $y(k)$ and $y_r(k)$ are the output and the reference input of the subsystem, respectively. $\xi(k)$ is an uncorrelated random series with zero mean representing the modeling error and process noise. Define the d-step ahead prediction error as

$$e_p(k+d) = y(k+d) - \hat{y}(k+d/k). \tag{3.10}$$

Substituting Eq. (3.10) into (3.9), and representing $\mathbf{A}(z^{-1})$, $\mathbf{B}(z^{-1})$ and $\mathbf{C}(z^{-1})$ as $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ for simplicity, we get

$$\mathbf{A}\; e_p(k) = \mathbf{B}\; y_r(k-d_0) - \mathbf{A}\; \hat{y}(k/k-d) + \mathbf{C}\; \xi(k). \tag{3.11}$$

Eq. (3.11) can be viewed as a new system, in which the input is the prediction $\hat{y}(k/k-d)$, the output is the prediction error $e_p(k)$, $y_r(k-d_0)$ is the measurable noise, and $\xi(k)$ is the unmeasurable noise. Define the cost function as $J = E\left\{e_p^2(k+d)\right\}$ and let

$$\mathbf{C} = \mathbf{E}_0\; \mathbf{A} + z^{-d_0}\; \mathbf{F},$$

where $\mathbf{E}_0$ and $\mathbf{F}$ are polynomials of z^{-1}, and $deg(\mathbf{E}_0) = d-1$, $deg(\mathbf{F}) = n-1$. By minimizing J, we get the optimal predictor

$$\hat{y}(k+d/k) = \frac{\mathbf{B}z^{-d_0}}{\mathbf{A}}\; y_r(k+d) + \frac{\mathbf{F}}{\mathbf{E}_0\; \mathbf{A}}\; e_p(k). \tag{3.12}$$

This is the prediction during the first iteration, that is, $\hat{y}^1(k+d/k) \equiv \hat{y}(k+d/k)$. The prediction error corresponding to Eq. (3.12) is

$$e_p(k+d) = \mathbf{E}_0\; \xi(k+d). \tag{3.13}$$

All the parameters of the system and the predictor are unknown and estimated on–line by an RLS (recursive least square) algorithm. $\hat{y}(k+d/k)$ is then computed using the estimated parameters.

In the knowledge–based controller, for each computed $y_r^i(k)$, the corresponding output prediction $\hat{y}^i(k+d/k)$ should be computed. Note that only $\hat{y}^0(k+d/k)$ is computed by Eq. (3.12), while giving $y_r^0(k) = y_d(k)$. The subsequent steps within this iteration are computed as

$$\hat{y}^i(k+d/k) = \hat{y}^0(k+d/k)\,\frac{y_r^i(k)}{y_d(k)}\,K_m^i, \qquad i = 1,\ 2,\ 3,\ \cdots, \tag{3.14}$$

where it is assumed that $y_d(k) \neq 0,\ \forall\ k \geq 0$. This formula is based on the assumption that the low–level, closed–loop subsystem has a linear response to its reference input. For the case of $i \geq 1$, using Eq. (3.14) instead of Eq. (3.12) not only simplifies the computation, but also reduces the sensitivity of the iteration to some estimated parameters. In Eq. (3.14), K_m^i is a gain factor. A servo controller is usually designed such that the system has a unity gain with respect to its reference input; it is reasonable to set $K_m^i = 1$. However, $\hat{y}^0(k+d/k)$ is computed with the error Eq. (3.13); in case the prediction error increases after the iterative operation, the gain factor would not necessarily be one for $i \geq 1$ so as to compensate for the prediction error.

Error Analysis

Thus far, we have assumed that the predictor gives the true output, which is not realistic. The effect of the prediction error on the tracking error is thus analyzed below. The tracking error is defined as

$$e_t(k) = y(k) - y_d(k). \tag{3.15}$$

As a result of the i-th iteration, suppose the actual reference input becomes $y_r^i(k)$

and the d-step ahead prediction of the output is $\hat{y}^i(k+d/k)$. $\hat{y}^0(k+d/k)$ is computed by the self–tuning predictor Eq. (3.12), and $\hat{y}^i(k+d/k)$, $i \geq 1$, is computed by Eq. (3.14). When

$$|e^i(k+d)| = |\hat{y}^i(k+d/k) - y_d(k+d)| \leq \epsilon, \tag{3.16}$$

the iterative learning process is stopped, and the result is given as the actual reference input $y_r^f(k)$ with the corresponding output prediction $\hat{y}^f(k+d/k)$. Because the iteratively computed prediction error is $e_p^i(k) = y(k) - \hat{y}^i(k/k-d)$, using Eq. (3.15) we get

$$e_t(k) - e_p^i(k) = \hat{y}^i(k/k-d) - y_d(k). \tag{3.17}$$

Eq. (3.16) can be used to convert Eq. (3.17) in the form $|e_t(k) - e_p^f(k)| \leq \epsilon$. Because $|e_t(k)| - |e_p^f(k)| \leq |e_t(k) - e_p^f(k)|$ and $|e_t(k) - e_p^f(k)| = |e_p^f(k) - e_t(k)|$, we conclude

$$|e_p^f(k)| - \epsilon \leq |e_t(k)| \leq |e_p^f(k)| + \epsilon. \tag{3.18}$$

This formula gives the upper and lower bounds of the tracking error when the knowledge–based controller is added. Specifically, it shows that the tracking error cannot be much less than the iteratively computed prediction error.

Obviously, an inaccurate prediction may degrade the performance of the knowledge–based controller. It can be seen from Eq. (3.13) that the prediction error of the self–tuning predictor is the moving average of a zero mean, uncorrelated random series of order $d-1$. Based on this observation, the subsequent steps of the prediction are iteratively computed by Eq. (3.14). To reduce the tracking error, a sophisticated predictor needs to be designed as a part of the iterative operation. The other way is to change the gain factor K_m^i in Eq. (3.14) in order to compensate for the prediction error.

Simulation Results

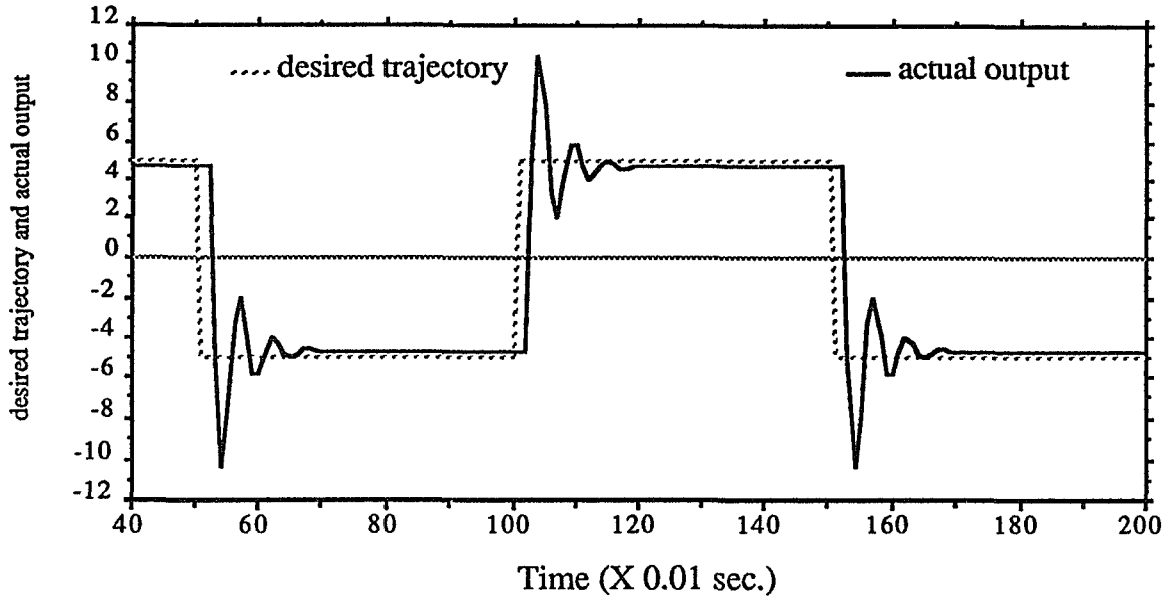


Figure 3.4: Output of a linear system without knowledge–based controller.

Simulations were carried out for two types of systems. First, we tested an open–loop linear system whose model is

$$y(k) = 0.45181\ y(k-1) + 0.47546\ y(k-2) - 0.04560\ u(k-1) - 0.00404\ u(k-2),$$

where y and u are the system output and control input, respectively. Using a proportional controller with $K_p = 20.9$, its closed–loop response is calculated and plotted in Fig. 3.4. The RMS (root mean square) tracking error is $RMSE = 2.13420$. By adding the knowledge–based controller, the performance is improved as shown in Fig. 3.5 with $RMSE = 1.30151$.

The second system we tested is a 2–link robot manipulator [AS86] — an open–loop nonlinear system:

$$\mathbf{H}(\boldsymbol{q})\ \ddot{\boldsymbol{q}} + \mathbf{C}(\boldsymbol{q},\ \dot{\boldsymbol{q}})\ \dot{\boldsymbol{q}} + \mathbf{G}(\boldsymbol{q}) = \boldsymbol{\tau}, \tag{3.19}$$

where $\boldsymbol{q} \equiv [q_1,\ q_2]^T$ and $\boldsymbol{\tau} \equiv [\tau_1,\ \tau_2]^T$ are the vectors of joint position and torque, respectively. $\mathbf{H}(\boldsymbol{q})$ is the inertia matrix, $\mathbf{C}(\boldsymbol{q},\ \dot{\boldsymbol{q}})\ \dot{\boldsymbol{q}}$ represents the Coriolis and centrifugal forces, and $\mathbf{G}(\boldsymbol{q})$ represents the gravitational force. Its configuration is

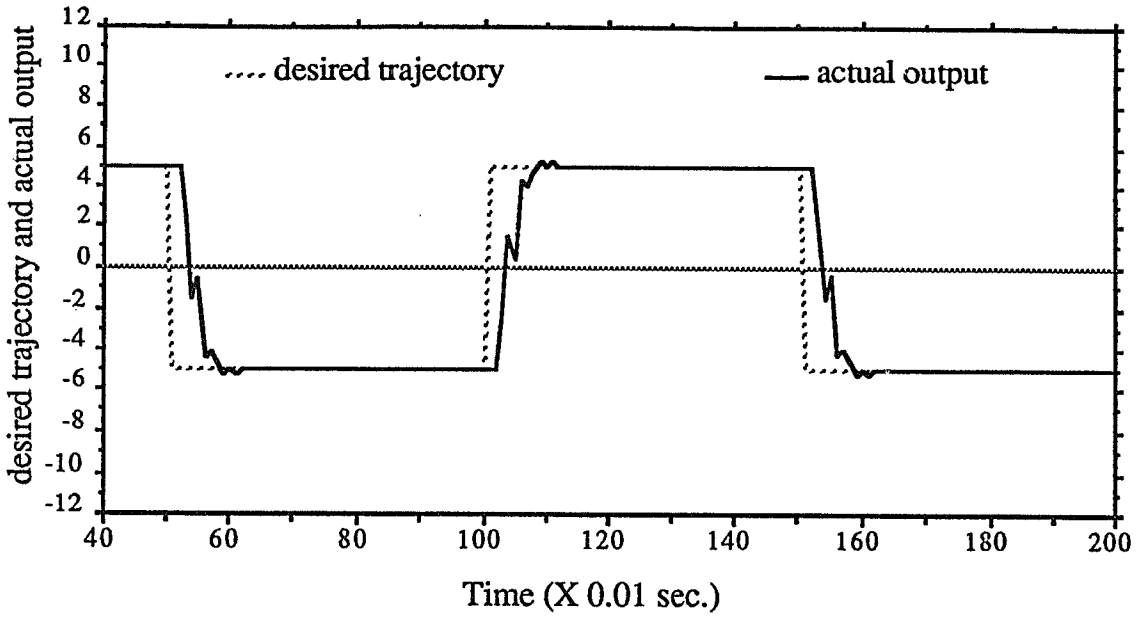


Figure 3.5: Output of a linear system with knowledge–based controller.

shown in Fig. 3.6, and dynamic and kinematic parameters are presented in Table 3.1. The detailed form of Eq. (3.19) is presented in the appendix of this dissertation. A

	Length	Mass center	Mass	Moment of inertia
Link 1	1 m	0.5 m	20 kg	0.8 $kg\ m\ s^2$
Link 2	1 m	0.5 m	10 kg	0.2 $kg\ m\ s^2$

Table 3.1: Kinematic and dynamic parameters of the simulated robot.

controller is designed with the computed torque algorithm [AS86]. The proportional and derivative gains are $K_p = 986.96$ and $K_D = 62.83$, which correspond to $\zeta = 1.0$ and $\omega_n = 10\ \pi/s$. The sampling interval is $T_s = 0.01sec.$ For simplicity, the desired trajectory is specified in joint space. To achieve the desired performance, the computed torque algorithm requires the accurate values of each term in Eq. (3.19). If inaccurate values of the inertia matrix (denoted by $\tilde{\mathbf{H}}$) are used in the computed torque algorithm, the performance is degraded as shown in Fig. 3.7 with an RMS

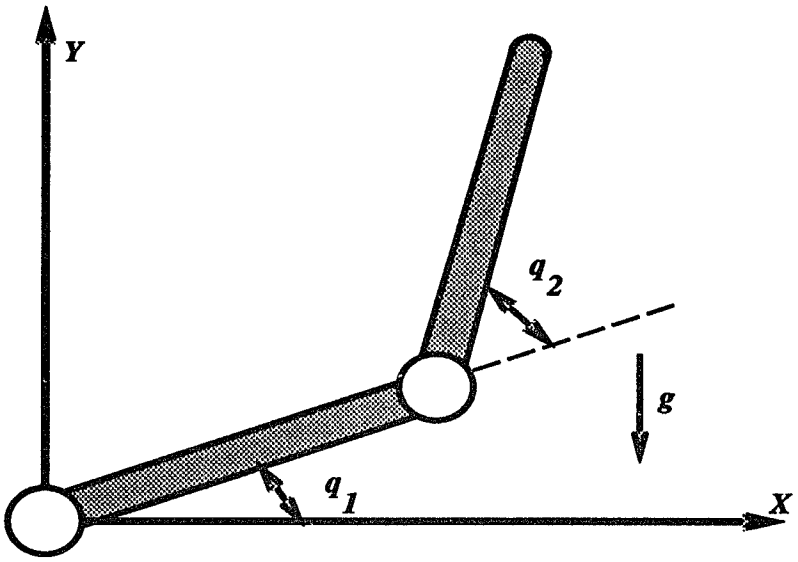


Figure 3.6: A 2–link robot manipulator.

tracking error $RMSE = 0.29228$. When the knowledge–based controller is added, the performance is improved as shown in Fig. 3.8 with $RMSE = 0.09654$. For comparison, with accurate values of the inertia matrix $\mathbf{H}$, the closed–loop response of the first link are plotted in Fig. 3.9 and 3.10. The corresponding RMS tracking errors are $RMSE = 0.23962$ without the knowledge–based controller and $RMSE = 0.04230$ with it. One can see that performance is also improved. Moreover, the simulation results of the second link are similar to those of the first link.

3.6 Summary

A knowledge–based controller has been proposed as a new architecture of IC and analyzed in detail. Its basic principle is to modify the reference input of the low–level subsystem in order to track a pre–designed trajectory accurately and to leave the internal structure and/or parameters of the subsystem unaffected. With the concept of iterative learning, the knowledge base is simple to design and the stability of the overall system is guaranteed. By using a d-step ahead predictor,

the undesirable effect of system time delay is eliminated and each reference input is analyzed in advance. This, in turn, solves the irrecoverable control problem. Now, the immediate work is to extend this scheme to the problem of coordinating multiple systems.

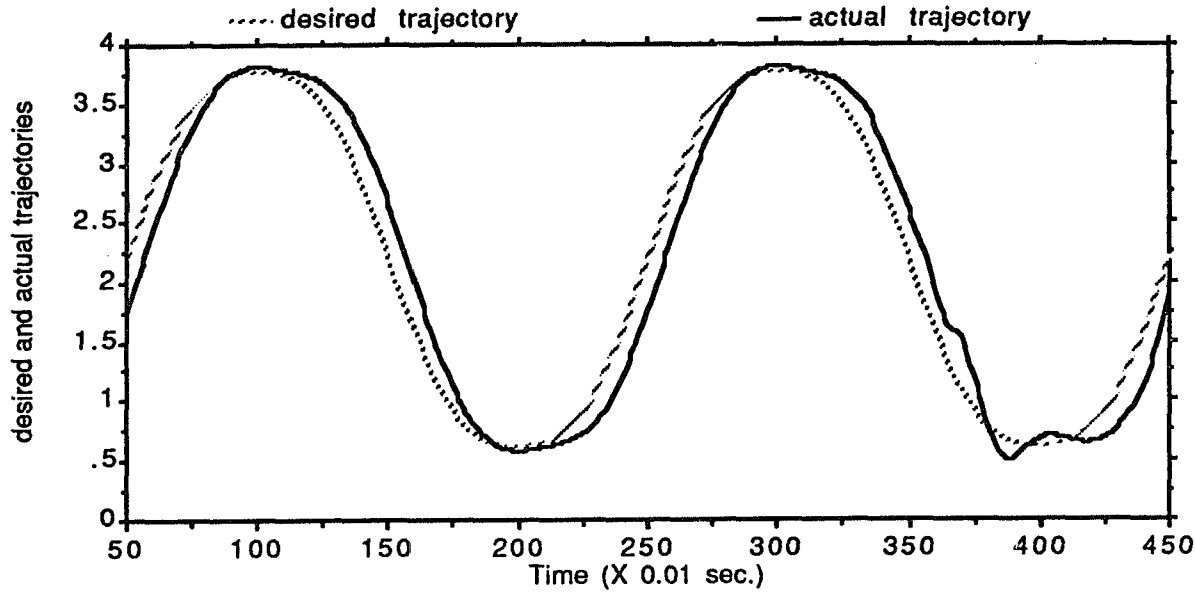


Figure 3.7: The position of joint 1 using $\tilde{\mathbf{H}}$ without knowledge-based controller.

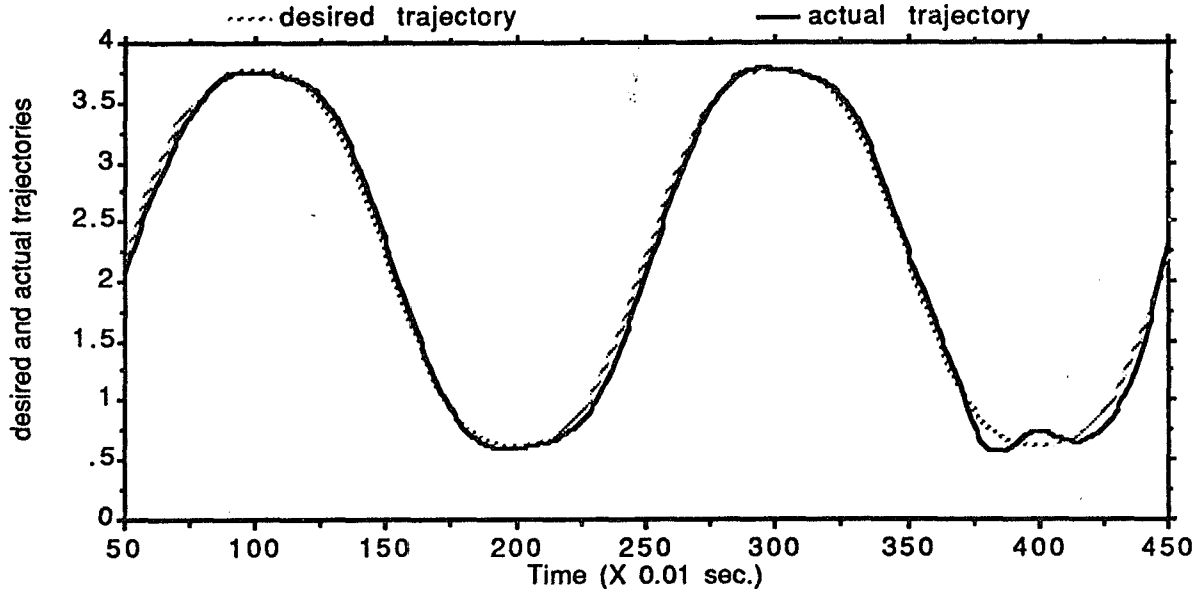


Figure 3.8: The position of joint 1 using $\tilde{\mathbf{H}}$ with knowledge-based controller.

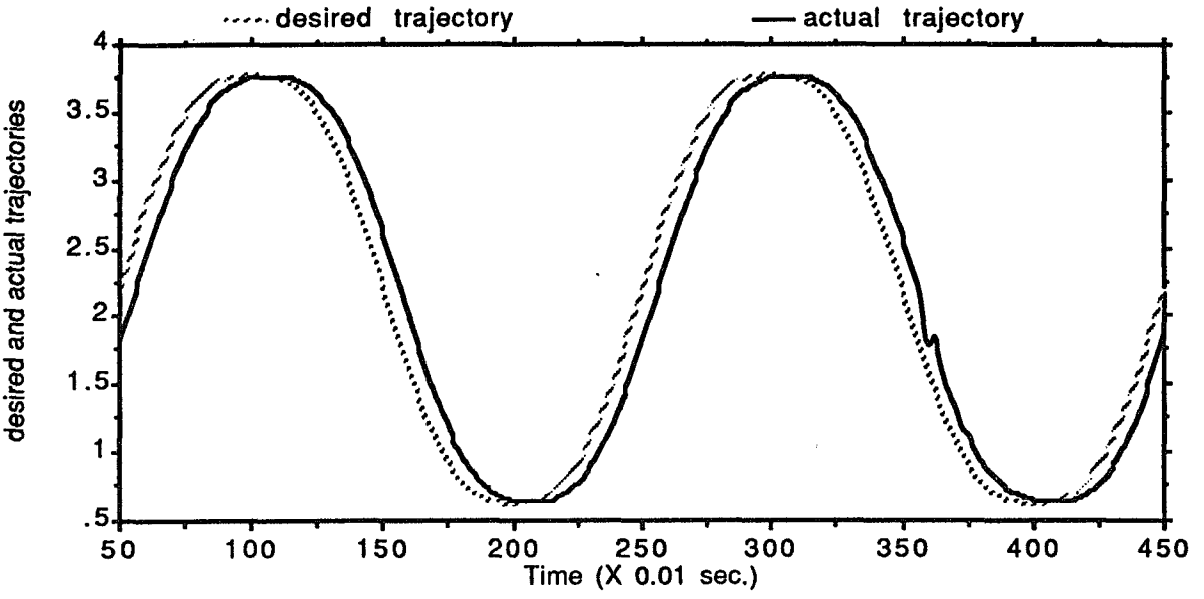


Figure 3.9: The position of joint 1 using **H** without knowledge–based controller.

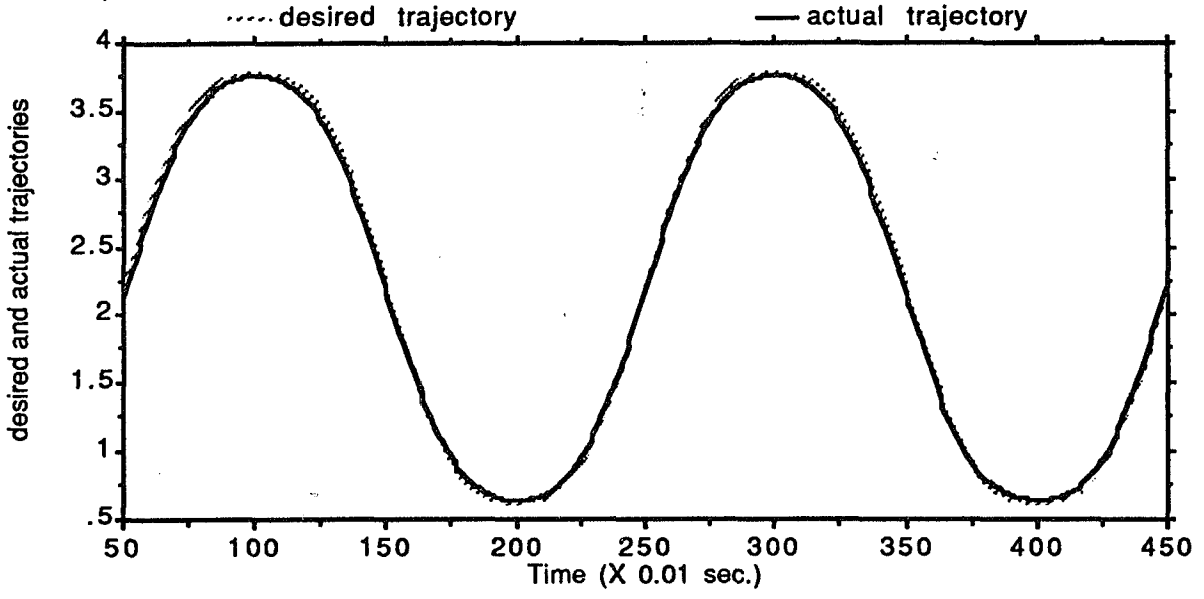


Figure 3.10: The position of joint 1 using **H** with knowledge–based controller.

CHAPTER IV

DESIGN OF A GENERAL–PURPOSE MIMO PREDICTOR WITH NEURAL NETWORKS

4.1 Introduction

The multi–step predictor is an essential part in the knowledge–based controller proposed in the previous chapter. In fact, there are numerous applications which require predicting system output in real time. For example, load forecasting in an electric power system is essential for an economical dispatch of electricity being generated. Automatic tracking of a flying object is the first step for fire control. In a power plant, prediction of the temperature and pressure at the outlet of a boiler is very useful to operators, especially during the period of startup and shutdown. Moreover, output prediction plays a vital role in predictive control. In this chapter, a general–purpose predictor is developed, which is not only for the knowledge–based controller but also for the applications mentioned above.

It is not difficult to design a predictor for linear SISO (single–input single–output) systems. For example, a self–tuning predictor is designed to predict the Si–content of pig iron for a blast–furnace [KC81]. However, it is a difficult task to design a multi–step predictor for an MIMO system. This is especially true for those systems with nonlinear, time–varying dynamics and/or long system time delays. The dynamic

equation of such a system is in general very difficult to derive, since the dynamic parameters are usually unknown, and/or even the internal structure of the system dynamics is sometimes unknown. No general method is known to exist for the design of a predictor for such systems.

Fortunately, NNs (neural networks) seem to shed light on solving this problem. Weigend *et al.* presented a good example of using NNs for time series prediction in which the NN was trained to predict the time serieses of the sunspot and a computational ecosystem [WHR90]. One of the major advantages of NNs is that they can represent any specified mapping with a learned configuration. A multilayer perceptron with a sufficient number of nodes is able to approximate *any* continuous mapping [Lip87, Bar89]. The output prediction of a system can be viewed as the mapping from the system's historical data and future inputs to future outputs, though it cannot usually be represented in an analytical form. The main intent of this chapter is to design a general-purpose MIMO predictor for a system using NNs.

One of the well-developed NNs is the multilayer perceptron with the back propagation training algorithm developed by [Wer88] and [RM86]. However, most NN applications are in the mode of train-first-and-then-operate; that is, the NN is trained with a set of training data before putting it in operation. After the NN becomes "well-trained", the weights of the NN will no longer be changed. This working mode is called the "*training-operation*" mode. For example, a multilayer perceptron was used in [BM89] as an SISO predictor to predict the pH value of a stirred tank reactor and this was then used as a basis for the design of a predictive controller. The moving-window concept was adopted and the weights of the NN would no longer be adjusted after the training period. The predictor proposed in [WHR90] also worked in the training-operation mode. However, this mode may not

be suitable for the control or prediction of some time–varying industrial processes. If adjusting NN's weights is viewed as a feedback from the NN's output error, then this feedback loop would be broken when the NN becomes "well–trained". (Obviously, this mode cannot be applied to time–varying systems.) To remedy this problem, an NN should be *updated*, rather than trained, that is, the weights of the NN should be adjusted on–line in order to keep track of the variation of a system. We call such a working mode as the "*updating*" mode. Updating an NN is essential to the design of an MIMO predictor for time–varying systems.

In the standard BP algorithm, the weights of the NN are adjusted by minimizing the network's output error. However, when an NN is used as a predictor, its output error is unknown since the future outputs of the predicted system are not known. We have designed a predictor which is updated only by using the historical data of the predicted system and does not require the knowledge of the the dynamic parameters nor the structure of the predicted system. Weights of the NN are dynamically adjusted to deal with the effects of nonlinear, time–varying properties, and/or long system–time delays. Because the NN–based predictor will always work in a closed loop, component failures in the NN will be learned and the NN will subsequently be re-configured, and then system reliability is improved. Furthermore, the parallel processing structure of the NN makes it suitable for high–dimensional systems.

To imitate an actual neuron, each node of an NN is usually designed to perform only scalar operations. However, for an MIMO system, if each node can only handle scalar operations, the size of NN may become too large to manage. We therefore propose to equip each node of the NN with the ability of vector operations in order to easily specify some known coupling relations within the predicted system and to get an easier (thus more intuitive) form of the training algorithm. This vector structured

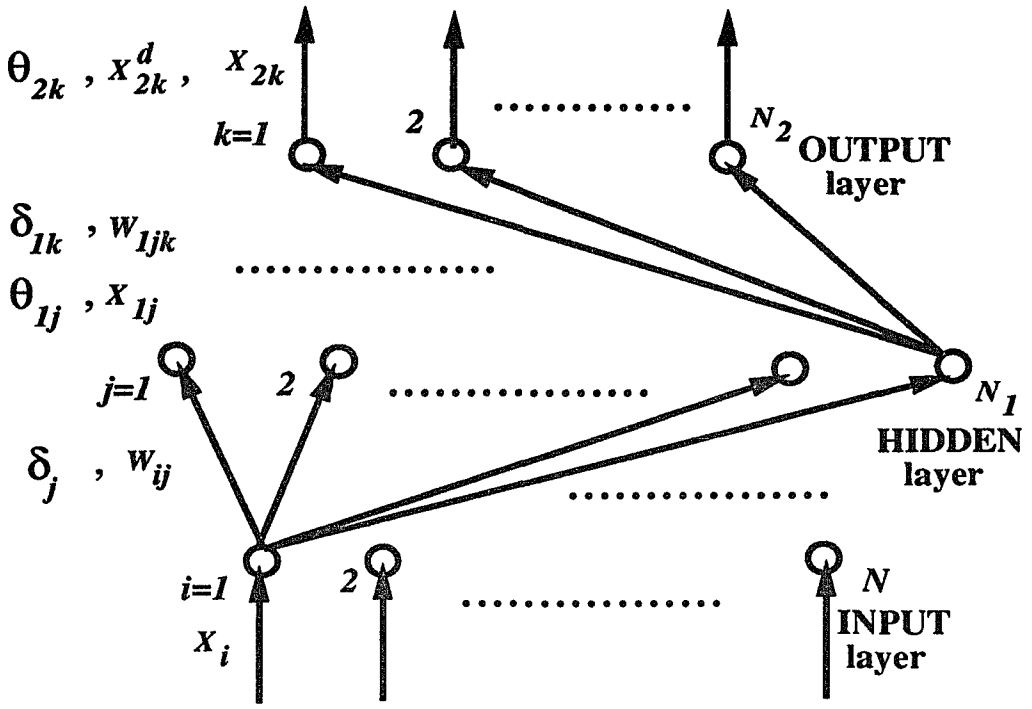


Figure 4.1: Basic structure of a three–layer perceptron.

multilayer perceptron will form the backbone of the proposed MIMO predictor.

This chapter is organized as follows. In Section 4.2, the basic structure of the NN–based predictor is described and the input–output mapping of an MIMO system is analyzed. Section 4.3 focuses on the problem of tracking time–varying systems by updating the NN. The scaling problem and error analysis of the NN–based predictor are also addressed in Section 4.3. A multi–dimensional BP (back propagation) algorithm is developed in Section 4.4; it is an extension of the scalar version presented in [RM86]. Simulation results are presented in Section 4.5 for various systems: MIMO linear, nonlinear, time–invariant, and time–varying. Finally, the main points of this chapter are summarized in Section 4.6.

4.2 Basic Structure of an MIMO NN–Based Predictor

The standard structure of a three–layer perceptron is shown in Fig. 4.1. With the

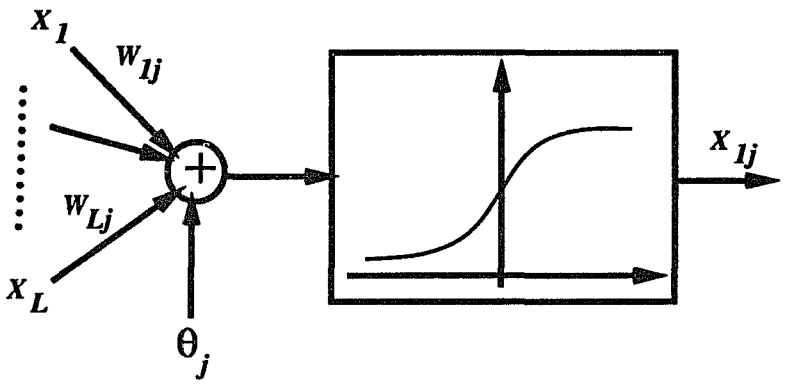


Figure 4.2: Basic structures of a neuron.

ability of learning from examples of a mathematical mapping, an NN can be trained to attain the dynamic property of the mapping. Typically, a set of input–output pairs $\{(u(k),\ y(k)) \mid y = G(u),\ k = 0,\ 1,\ 2,\ \cdots\}$ are used as *training data*, where G is the input–output mapping. These training data are used to adjust the weights of the NN which represents the input–output mapping. The BP algorithm attempts to approximate a mapping in the sense of least mean squares [Lip87]. The computation of an NN with the BP algorithm consists of two steps: computing the output of the NN forward from the INPUT layer to the OUTPUT layer, and adjusting the weights backward from the OUTPUT layer to the INPUT layer. The computation at each node is shown in Fig. 4.2 with $X_j = f(\sum_{i=1}^{L} W_{ij}\ X_i + \theta_j)$, where X_j is the output of node j, X_i the input from node i, W_{ij} the weight on the arc from node i to node j, θ_j the threshold at node j, and $f(x) \equiv \dfrac{1}{1+\exp(-x)}$ is a sigmoid function. Once new output at the OUTPUT layer is ready and denoted by X_{2k}, then the output error of the NN is computed as $E_k = X_{2k}^d - X_{2k}$, where X_{2k}^d is the desired output of the NN. The weights of each layer are then modified according to the output error E_k. In other words, this training process is driven by the output error of the NN. It has been proved that a multilayer perceptron with two hidden layers and a sufficient

number of hidden nodes can approximate *any* input–output mapping [Cyb89].

If the predicted system is causal, then the dynamic relationship between input and output can be conceptually represented as

$$\boldsymbol{Y}(k) = \mathbf{h}(\boldsymbol{Y}(k-i),\ \boldsymbol{U}(k-j),\ k), \tag{4.1}$$

where $\boldsymbol{Y} \in \mathbf{R}^p$ is the output vector, $\boldsymbol{U} \in \mathbf{R}^n$ the input vector, $k \in \mathbf{Z}$ the discrete time index, and $\mathbf{h} : \mathbf{R}^p \times \mathbf{R}^n \times \mathbf{Z} \rightarrow \mathbf{R}^p$. $\boldsymbol{Y}(k-i)$, $i = 1,\ 2,\ \cdots$ is the historical data of the system output. $\boldsymbol{U}(k-j)$, $j = 0,\ 1,\ 2,\ \cdots$ is the system input at and before time k. If the mapping, $\mathbf{h}$, in Eq. (4.1) were known, the future output of the system could be computed step by step by predicting the future inputs $\boldsymbol{U}(k+d)$, $d = 1,\ 2,\ \cdots$. However, it is usually very difficult to derive a closed–form expression for Eq. (4.1) — even if such a closed–form existed. The d-step ahead prediction of $\boldsymbol{Y}$ can be represented as

$$\begin{aligned} \hat{\boldsymbol{Y}}(k+d/k) &= \mathbf{h}_p(\bar{\mathbf{Y}},\ \bar{\mathbf{U}},\ k), \qquad (4.2)\\ \text{where}\quad \bar{\mathbf{Y}} &= [\boldsymbol{Y}(k),\ \boldsymbol{Y}(k-1),\ \cdots,\ \boldsymbol{Y}(k-i)] \\ \bar{\mathbf{U}} &= [\boldsymbol{U}(k+i_1),\ \cdots,\ \boldsymbol{U}(k),\ \boldsymbol{U}(k-1),\ \cdots,\ \boldsymbol{U}(k-i_2)] \\ & \mathbf{h}_p\colon \mathbf{R}^p \times \mathbf{R}^n \times \mathbf{Z} \rightarrow \mathbf{R}^p, \end{aligned}$$

where i, i_1 and i_2 are positive integers. The parameters and/or structure of this mapping are not known either. It is therefore practically impossible to design a general–purpose predictor with mathematical synthesis alone, though a closed–form expression for such an MIMO mapping may sometimes exist. So, we propose to design an NN which will learn the mapping Eq. (4.2). A three–layer perceptron is used for this purpose, where the inputs are $\bar{\mathbf{Y}}$ and $\bar{\mathbf{U}}$, and the output is $\hat{\boldsymbol{Y}}(k+d/k)$, $d = 1,\ 2,\ \cdots$. There are two major difficulties in implementing this scheme.

The first problem is related to the training process. In the standard BP algorithm, the NN should be trained by minimizing the NN's output errors. For our case, however, the desired values of the NN's output are the system outputs, $\boldsymbol{Y}(k+d)$, $d = 1, 2, \cdots$. The network–output errors are then the system prediction errors which are computed as

$$\boldsymbol{e}_p(k+d) = \boldsymbol{Y}(k+d) - \hat{\boldsymbol{Y}}(k+d/k).$$

This implies that the network should be trained by using the system's unknown future outputs $\boldsymbol{Y}(k+d)$. A set of training data can be acquired beforehand, and used to train the NN. After the NN is "well trained", the NN will no longer modify its weights and produce the output, while the inputs are present at the INPUT nodes. For time–varying mappings, however, it is meaningless to say that an NN is "well trained." Moreover, as pointed out in the previous section, this training–operation mode implies that the NN work in an open loop after the training, which is not acceptable in a real–time control or prediction system. Therefore, our MIMO predictor needs an NN that is updated on–line in order to keep track of a time–varying mapping and to work always in a closed loop. More on this will be discussed in the next section.

The second problem is how to efficiently and clearly represent the training algorithm for an MIMO system. Note that Eq. (4.2) is an MIMO mapping. If each node of the NN can only handle scalar operations, then we need a fully–connected multi-layer mesh. Each node at the INPUT or OUTPUT layer of this mesh corresponds to an element of the input or output vector of the mapping. By using the standard BP algorithm, it is difficult to express some known coupling relations within the mapping and obtain a set of succinct formulas for the training algorithm. Therefore, we need an NN which can handle vector operations in order to reduce the total number of

the nodes required. Such an NN will be discussed in Section 4.4.

4.3 Tracking a Time–Varying System and Error Analysis

4.3.1 Training Algorithm for the Updating Mode

Suppose the predicted system is an SISO system with output $y(t)$ and its d–step ahead prediction $\hat{y}(t+d/t)$ at time t (t is the continuous–time index). Let $X_{2k}(t)$ and $X_{2k}^d(t)$ be respectively the actual output of the NN–based predictor and its desired value at time t. Then the network–output error is computed by

$$E_k(t) = X_{2k}^d(t) - X_{2k}(t).$$

In the standard BP algorithm, we must use this network–output error to train the NN. However, when the NN is used as a predictor, the network's output is the prediction of the system output,[1] and the network–output error is the prediction error:

$$E_k(t) = X_{2k}^d(t) - X_{2k}(t) = y(t+d) - \hat{y}(t+d/t). \tag{4.3}$$

$E_k(t)$ is unavailable since the system's future output $y(t+d)$ is not available at time t. Hence, we must use the system's historical data to update the NN–based predictor on–line in order to maintain the closed–loop operation by keeping track of a time–varying system. To update the NN–based predictor, instead of using Eq. (4.3), we propose to use a posterior prediction error:

$$E_k(t-d) = X_{2k}^d(t-d) - X_{2k}(t-d) = y(t) - \hat{y}(t/t-d).$$

This arrangement is equivalent to cascading the NN with delay elements, as shown in Fig. 4.3. In what follows, a modified BP algorithm is derived to handle these

[1] In fact, the network's output X_{2k} is a scaled value of the system's output prediction $\hat{y}$. At this stage, it is assumed that $\hat{y}$ is within the range of (0, 1). The scaling problem will be discussed later.

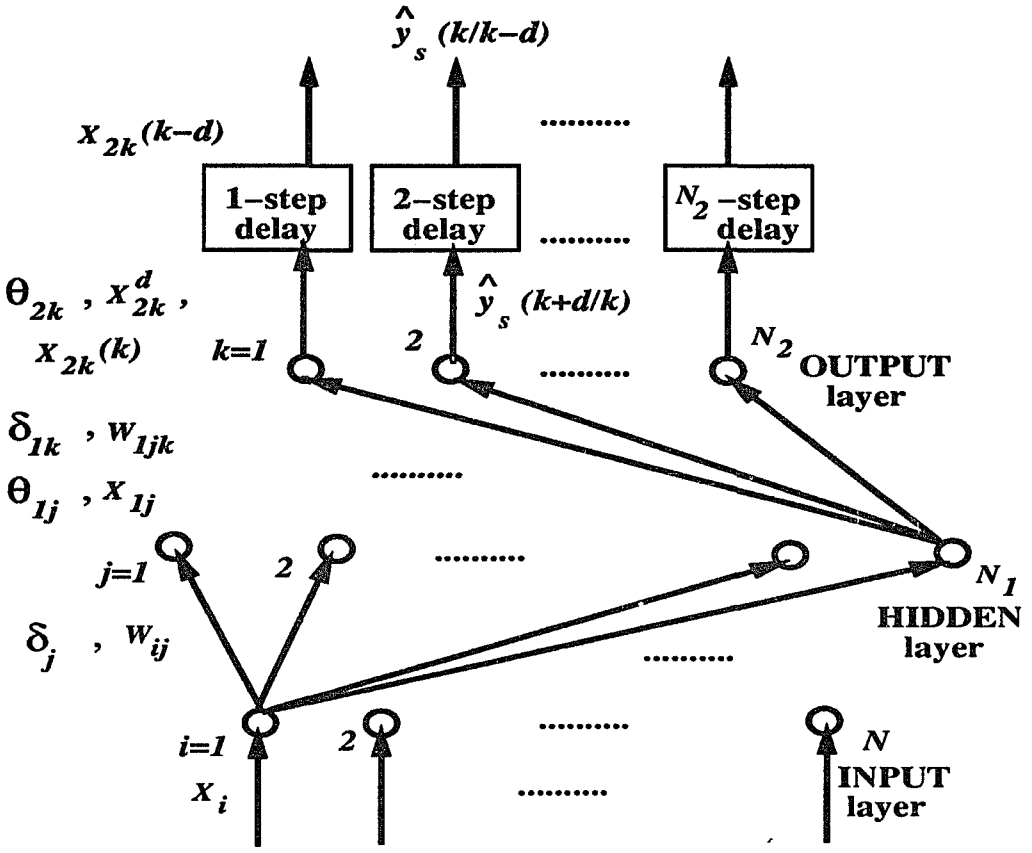


Figure 4.3: Basic structures of the NN–based multi–step predictor.

delay elements. The formulas are given for SISO systems, which will be extended to the MIMO case in the next section.

1. Compute the Output of the HIDDEN Layer, X_{1j}

The HIDDEN layer's outputs are

$$X_{1j}(t) = \frac{1}{1+\exp\left(-O_{1j}-\theta_{1j}\right)},$$

$$\text{where} \quad O_{1j} = \sum_{i=1}^{N} W_{ij}\, X_i(t), \qquad j = 1,\ 2,\ \cdots,\ N_1,$$

X_i is the input at INPUT node i, W_{ij} is the weight from INPUT node i to HIDDEN node j, and θ_{1j} is the threshold at HIDDEN node j.

2. Compute the Output of the OUTPUT Layer, X_{2k}

The outputs of the OUTPUT layer are

$$X_{2k}(t) = \frac{1}{1+\exp\left(-O_{2k}-\theta_{2k}\right)}, \tag{4.4}$$
$$\text{where} \quad O_{2k} = \sum_{j=1}^{N_1} W_{1jk}\, X_{1j}(t), \qquad k = 1,\ 2,\ \cdots,\ N_2,$$

W_{1jk} is the weight from HIDDEN node j to OUTPUT node k, and θ_{2k} is the threshold at OUTPUT node k.

3. Update the Weights from the HIDDEN to OUTPUT Layer, W_{1jk}

We want to use the delayed data $X_{2k}(t-d)$ to update the network. The cost function of the network is defined by

$$E(t) = \frac{1}{2}\sum_{k=1}^{N_2}\left(E_k(t-d)\right)^2 = \frac{1}{2}\sum_{k=1}^{N_2}\left(X_{2k}^d(t-d) - X_{2k}(t-d)\right)^2. \tag{4.5}$$

Let the updated weights be $W_{1jk}(t+\Delta t) = W_{1jk}(t) + \Delta W_{1jk}$. Using the gradient algorithm,

$$\Delta W_{1jk} \propto -\frac{\partial E(t)}{\partial W_{1jk}} = -\frac{\partial E(t)}{\partial O_{2k}}\,\frac{\partial O_{2k}}{\partial W_{1jk}}. \tag{4.6}$$

From Eq. (4.4), we have

$$\frac{\partial E(t)}{\partial O_{2k}} = \frac{\partial E(t)}{\partial X_{2k}(t)}\,\frac{\partial X_{2k}(t)}{\partial O_{2k}} = \frac{\partial E(t)}{\partial X_{2k}(t)}\,X_{2k}(t)\,\left(1-X_{2k}(t)\right), \quad \text{and}$$
$$\frac{\partial O_{2k}}{\partial W_{1jk}} = \frac{\partial}{\partial W_{1jk}}\left(\sum_{l=1}^{N_1} W_{1lk}\, X_{1l}(t)\right) = X_{1j}(t).$$

Then, we get

$$\frac{\partial E(t)}{\partial W_{1jk}} = \frac{\partial E(t)}{\partial X_{2k}(t)}\,X_{2k}(t)\,\left(1-X_{2k}(t)\right)\,X_{1j}(t). \tag{4.7}$$

Because

$$\begin{aligned}
\frac{\partial E(t)}{\partial X_{2k}(t)} &= \frac{\partial E(t)}{\partial X_{2k}(t-d)}\,\frac{\partial X_{2k}(t-d)}{\partial X_{2k}(t)} \\
&= \frac{1}{2}\,\frac{\partial}{\partial X_{2k}(t-d)}\left(\sum_{l=1}^{N_2}\left(X_{2l}^d(t-d) - X_{2l}(t-d)\right)^2\right)\frac{\partial X_{2k}(t-d)}{\partial X_{2k}(t)} \\
&= -\left(X_{2k}^d(t-d) - X_{2k}(t-d)\right)\frac{\partial X_{2k}(t-k)}{\partial X_{2k}(t)}.
\end{aligned}$$

Let

$$\delta_{1k} = \left(X_{2k}^{d}(t-d) - X_{2k}(t-d)\right)\ X_{2k}(t)\ (1 - X_{2k}(t))\,, \tag{4.8}$$

then Eq. (4.7) becomes

$$\frac{\partial E(t)}{\partial W_{1jk}} = -\ \delta_{1k}\ X_{1j}(t)\ \frac{\partial X_{2k}(t-d)}{\partial X_{2k}(t)}. \tag{4.9}$$

Now, the problem is how to compute $\dfrac{\partial X_{2k}(t-d)}{\partial X_{2k}(t)}$. Because $X_{2k}(t) = \hat{y}(t+d/t)$ is the d–step ahead prediction and $X_{2k}(t-d) = \hat{y}(t/t-d)$ is the same value of $\hat{y}(t+d/t)$ delayed by d, we conclude

$$\frac{\partial X_{2k}(t-d)}{\partial X_{2k}(t)} = 1, \quad \forall\ t\ \text{ and }\ d. \tag{4.10}$$

In Eq. (4.6), we set $\Delta W_{1jk} = -\ \eta_1 \dfrac{\partial E(t)}{\partial W_{1jk}}$, $\eta_1 > 0$ is a gain factor. Therefore, using the results of Eqs. (4.9) and (4.10), we get

$$\begin{aligned} W_{1jk}(t+\Delta t) &= W_{1jk}(t) + \Delta W_{1jk} \\ \Delta W_{1jk} &= \eta_1\ \delta_{1k}\ X_{1j}(t), \end{aligned}$$

where δ_{1k} is given by Eq. (4.8). This has the same form as the standard BP algorithm, but the definition of the cost function in Eq. (4.5) is different. Similarly to the above process, other formulas are listed below without any detailed account.

4. Update the Weights from the INPUT to HIDDEN Layer, W_{ij}

$$\begin{aligned} W_{ij}(t+\Delta t) &= W_{ij}(t) + \eta\ \delta_j\ X_i(t), \\ \text{where}\quad \delta_j &= \left[\sum_{k=1}^{N_2} \delta_{1k}\ W_{1jk}\right]\ X_{1j}(t)\ (1 - X_{1j}(t))\,, \end{aligned} \tag{4.11}$$

where $\eta > 0$ is a gain factor.

5. Update the Thresholds θ_{2k} and θ_{1j}

$$\begin{aligned} \theta_{2k}(t+\Delta t) &= \theta_{2k}(t) + \eta_{1\theta}\ \delta_{1k}, \\ \theta_{1j}(t+\Delta t) &= \theta_{1j}(t) + \eta_{\theta}\ \delta_j, \end{aligned}$$

where $\eta_{1\theta} > 0$ and $\eta_{\theta} > 0$ are gain factors, and δ_{1k}, δ_j are given by Eq. (4.8) and Eq. (4.11), respectively.

4.3.2 Scaling Problem and Error Analysis

In the NN, the output of each node passes through a sigmoid function (Fig. 4.2) which forces the output of each node to be within the range of (0, 1), and thus, the inputs to the INPUT nodes also need to be scaled to this range. Suppose the output of a predicted system is $y(k) \in (y_{min},\ y_{max})$ and a linear scaling formula is used, then the scaled value of $y(k)$ is given as

$$y_s(k) = \frac{y(k) - y_{min}}{y_{max} - y_{min}}. \tag{4.12}$$

The output of the NN–based predictor is the scaled value of d–step ahead prediction $X_{2k}(k) = \hat{y}_s(k + d/k)$. Then the unscaled values are

$$y(k) = y_s(k)\ (y_{max} - y_{min}) + y_{min} \tag{4.13}$$

$$\hat{y}(k + d/k) = \hat{y}_s(k + d/k)\ (y_{max} - y_{min}) + y_{min}. \tag{4.14}$$

From Eqs. (4.13) and (4.14), we get the absolute prediction error

$$e_p(k) = y(k) - \hat{y}(k/k - d) = e_{ps}(k)\ (y_{max} - y_{min}), \tag{4.15}$$

$$\text{where}\quad e_{ps}(k) = y_s(k) - \hat{y}_s(k/k - d) = X^d_{2k}(k - d) - X_{2k}(k - d)$$

is the absolute output error of the NN–based predictor.

The relative prediction error is defined as

$$\Delta e_p(k) \equiv \left| \frac{y(k) - \hat{y}(k/k - d)}{y_{max} - y_{min}} \right| = \left| \frac{e_p(k)}{y_{max} - y_{min}} \right|.$$

Similarly, the relative output error of the NN is defined as

$$\Delta e_{ps}(k) \equiv \left| \frac{y_s(k) - \hat{y}_s(k/k - d)}{1.0} \right|. \tag{4.16}$$

Substituting the scaling formula Eq. (4.12) into Eq. (4.16), we get

$$\Delta e_{ps}(k) = \left| \frac{y(k) - \hat{y}(k/k-d)}{y_{max} - y_{min}} \right| = \Delta e_p(k). \tag{4.17}$$

From Eqs. (4.15) and (4.17), we can observe that the unscaled absolute prediction error may be much larger than the scaled one, but the unscaled relative prediction error is the same as the scaled one. This indicates that the accuracy of the NN–based predictor depends only on the accuracy of the NN's approximation to the actual mapping. Finally, y_{max} and y_{min} should be determined as

$$y_{max} = \sup \{y(k) \mid \text{for all } k \geq 0\}, \quad y_{min} = \inf \{y(k) \mid \text{for all } k \geq 0\},$$

such that the range of the scaled value approaches (0, 1) in order to excite the network.

4.4 Multi–Dimensional Back Propagation Algorithm

Each node within a conventional NN is designed to handle only scalar operations, and the number of the nodes or layers is increased to deal with a more complex I/O mapping. However, for a complex MIMO mapping like Eq. (4.2), the network size may become too large to manage. Alternatively, one can equip each node of an NN with the ability of vector operations, because this is easier to express some known coupling relations and will result in a set of succinct formulas. So, all inputs and outputs of this NN are vectors. Referring to Fig. 4.1, let $\boldsymbol{X}_i \in \mathbf{R}^n$, $\boldsymbol{X}_{1j} \in \mathbf{R}^m$, and $\boldsymbol{X}_{2k} \in \mathbf{R}^p$ be the output of the INPUT, HIDDEN and OUTPUT layer, respectively, for $1 \leq i \leq N$, $1 \leq j \leq N_1$, and $1 \leq k \leq N_2$. Extension of the BP algorithm to a vector form is presented below.

1. Computing the output of the HIDDEN layer $\boldsymbol{X}_{1j}$

The output of the HIDDEN layer is computed by

$$\begin{aligned} \boldsymbol{X}_{1j} &\equiv [x_{1j1}, \cdots, x_{1jm}]^T = \boldsymbol{f}_j(\boldsymbol{O}_{1j}) \\ &= \left[\frac{1}{1+\exp(-o_{1j1}-\theta_{1j1})}, \cdots, \frac{1}{1+\exp(-o_{1jm}-\theta_{1jm})}\right]^T, \qquad (4.18) \\ \boldsymbol{O}_{1j} &= \sum_{i=1}^{N} \mathbf{W}_{ij}\, \boldsymbol{X}_i, \quad j=1,\ 2,\ \cdots,\ N_1, \qquad (4.19) \end{aligned}$$

where $\mathbf{W}_{ij} \in \mathbf{R}^{m\times n}$ is the weighting matrix from node i of the INPUT layer to node j of the HIDDEN layer, $\boldsymbol{f}_j : \mathbf{R}^m \rightarrow \mathbf{R}^m$ is defined as a sigmoid function of each component of a vector, and $\Theta_{1j} \equiv [\theta_{1j1}, \cdots, \theta_{1jm}]^T$ is the threshold vector at node j of the HIDDEN layer.

2. Computing the output of the OUTPUT layer $\boldsymbol{X}_{2k}$

Similarly, the output of the OUTPUT layer is

$$\begin{aligned} \boldsymbol{X}_{2k} &\equiv [x_{2k1}, \cdots, x_{2kp}]^T = \boldsymbol{f}_k(\boldsymbol{O}_{2k}) \\ &= \left[\frac{1}{1+\exp(-o_{2k1}-\theta_{2k1})}, \cdots, \frac{1}{1+\exp(-o_{2kp}-\theta_{2kp})}\right]^T, \qquad (4.20) \\ \boldsymbol{O}_{2k} &= \sum_{j=1}^{N_1} \mathbf{W}_{1jk}\, \boldsymbol{X}_{1j}, \quad k=1,\ 2,\ \cdots,\ N_2, \qquad (4.21) \end{aligned}$$

where $\mathbf{W}_{1jk} \in \mathbf{R}^{p\times m}$ is the weighting matrix from node j of the HIDDEN layer to node k of the OUTPUT layer, $\boldsymbol{f}_k : \mathbf{R}^p \rightarrow \mathbf{R}^p$ is defined as a sigmoid function of each component of a vector, and $\Theta_{2k} \equiv [\theta_{2k1}, \cdots, \theta_{2kp}]^T$ is the threshold vector at node k of the OUTPUT layer. Note that if $m = p = n$, $\mathbf{W}_{ij} = diag\,[w_{11}, \cdots, w_{nn}]_{ij}$ and $\mathbf{W}_{1jk} = diag\,[w_{111}, \cdots, w_{1nn}]_{jk}$, then the system is uncoupled.

3. Updating the Weights from the HIDDEN to OUTPUT Layer $\mathbf{W}_{1jk}$

Let the desired output of the network be $\boldsymbol{X}_{2k}^d = \left[x_{2k1}^d, \cdots, x_{2kp}^d\right]^T$, and let the output error of node k be defined as $\boldsymbol{E}_k = \boldsymbol{X}_{2k}^d - \boldsymbol{X}_{2k}$ and the cost function be defined as

$$E = \frac{1}{2}\sum_{k=1}^{N_2} \boldsymbol{E}_k^T\, \boldsymbol{E}_k = \frac{1}{2}\sum_{k=1}^{N_2} (\boldsymbol{X}_{2k}^d - \boldsymbol{X}_{2k})^T\, (\boldsymbol{X}_{2k}^d - \boldsymbol{X}_{2k}). \qquad (4.22)$$

We want to update $\mathbf{W}_{1jk}$ and $\mathbf{W}_{ij}$ by minimizing E and taking the form of

$$\mathbf{W}_{1jk}(t+\Delta t) = \mathbf{W}_{1jk}(t) + \Delta \mathbf{W}_{1jk}, \tag{4.23}$$

$$\mathbf{W}_{ij}(t+\Delta t) = \mathbf{W}_{ij}(t) + \Delta \mathbf{W}_{ij}, \tag{4.24}$$

where t is the continuous time index.

By using the gradient algorithm, we should set

$$\Delta \mathbf{W}_{1jk} \propto -\left[\frac{\partial E}{\partial \mathbf{W}_{1jk}}\right]^T = -\left[\frac{\partial E}{\partial \boldsymbol{O}_{2k}}\, \mathrm{T}_1\right]^T, \tag{4.25}$$

where $\dfrac{\partial E}{\partial \boldsymbol{O}_{2k}} \in \mathbf{R}^{1\times p}$, and $\mathrm{T}_1 \equiv \dfrac{\partial \boldsymbol{O}_{2k}}{\partial \mathbf{W}_{1jk}} \in \mathbf{R}^{p\times m\times p}$ is a three dimensional tensor since $\boldsymbol{O}_{2k} \in \mathbf{R}^{p\times 1}$ and $\mathbf{W}_{1jk} \in \mathbf{R}^{p\times m}$.

To compute $\dfrac{\partial E}{\partial \boldsymbol{O}_{2k}}$, referring to Eq. (4.22), we get

$$\frac{\partial E}{\partial \boldsymbol{O}_{2k}} = \frac{\partial E}{\partial \boldsymbol{X}_{2k}}\, \frac{\partial \boldsymbol{X}_{2k}}{\partial \boldsymbol{O}_{2k}} = -(\boldsymbol{X}^d_{2k} - \boldsymbol{X}_{2k})^T\, \frac{\partial \boldsymbol{X}_{2k}}{\partial \boldsymbol{O}_{2k}}. \tag{4.26}$$

From Eq. (4.20), we obtain

$$\frac{\partial \boldsymbol{X}_{2k}}{\partial \boldsymbol{O}_{2k}} = \frac{\partial\, \boldsymbol{f}_k(\boldsymbol{O}_{2k})}{\partial \boldsymbol{O}_{2k}} = \begin{bmatrix} \dfrac{\partial x_{2k1}}{\partial o_{2k1}} & \cdots & \dfrac{\partial x_{2k1}}{\partial o_{2kp}} \\ \cdots & \cdots & \cdots \\ \dfrac{\partial x_{2kp}}{\partial o_{2k1}} & \cdots & \dfrac{\partial x_{2kp}}{\partial o_{2kp}} \end{bmatrix}. \tag{4.27}$$

Because $x_{2kl} = \dfrac{1}{1+\exp\left(-o_{2kl}-\theta_{2kl}\right)}$ and $\dfrac{\partial x_{2kl}}{\partial o_{2kl}} = x_{2kl}\,(1-x_{2kl})$, Eq. (4.27) can be written in the form

$$\frac{\partial \boldsymbol{X}_{2k}}{\partial \boldsymbol{O}_{2k}} = diag\left[x_{2k1}(1-x_{2k1}),\ \cdots,\ x_{2kp}(1-x_{2kp})\right]. \tag{4.28}$$

Therefore, substituting Eq. (4.28) into (4.26) and using the notation $\boldsymbol{\delta}_{1k}$, we get

$$\begin{aligned} \boldsymbol{\delta}_{1k} &\equiv -\frac{\partial E}{\partial \boldsymbol{O}_{2k}} = (\boldsymbol{X}^d_{2k} - \boldsymbol{X}_{2k})^T\, \frac{\partial \boldsymbol{X}_{2k}}{\partial \boldsymbol{O}_{2k}} \\ &= (\boldsymbol{X}^d_{2k} - \boldsymbol{X}_{2k})^T\, diag\left[x_{2k1}(1-x_{2k1}),\ \cdots,\ x_{2kp}(1-x_{2kp})\right] \in \mathbf{R}^{1\times p}. \end{aligned} \tag{4.29}$$

To compute T_1, from Eq. (4.21), we have the l-th component of $\boldsymbol{O}_{2k}$ as $o_{2kl} = \sum_{j=1}^{N_1} (\boldsymbol{W}_l)_{1jk}\ \boldsymbol{X}_{1j}$, $l = 1,\ 2,\ \cdots,\ p$, where $(\boldsymbol{W}_l)_{1jk}$ is the l-th row of $\mathbf{W}_{1jk}$. Then, the l-th matrix of T_1, $\mathbf{T}_{1l} \equiv \dfrac{\partial o_{2kl}}{\partial \mathbf{W}_{1jk}} \in \mathbf{R}^{m \times p}$, $l = 1,\ 2,\ \cdots,\ p$, has the form of

$$\mathbf{T}_{1l} \equiv \frac{\partial o_{2kl}}{\partial \mathbf{W}_{1jk}} = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \dfrac{\partial}{\partial (\boldsymbol{W}_l)_{1jk}^T} \left(\sum_{q=1}^{N_1} (\boldsymbol{W}_l)_{1qk}\ \boldsymbol{X}_{1q} \right) \\ \mathbf{0} \\ \vdots \end{bmatrix}^T \tag{4.30}$$

$$= \begin{bmatrix} \mathbf{0} \\ \vdots \\ \dfrac{\partial}{\partial (\boldsymbol{W}_l)_{1jk}^T} \left((\boldsymbol{W}_l)_{1jk}\ \boldsymbol{X}_{1j} \right) \\ \mathbf{0} \\ \vdots \end{bmatrix}^T = \begin{bmatrix} \mathbf{0} \\ \vdots \\ (\boldsymbol{X}_{1j})^T \\ \mathbf{0} \\ \vdots \end{bmatrix}^T \longleftarrow \text{at the } l\text{-th row.}$$

In Eq. (4.25), we set $\Delta \mathbf{W}_{1jk} = -\ \eta_1 \left[\dfrac{\partial E}{\partial \mathbf{W}_{1jk}} \right]^T$ where $\eta_1 > 0$ is the gain factor of the OUTPUT layer. Therefore, from Eqs. (4.29) and (4.30), we get

$$\Delta \mathbf{W}_{1jk} = -\ \eta_1 \left[\frac{\partial E}{\partial \boldsymbol{O}_{2k}}\ \mathrm{T}_1 \right]^T = \eta_1 \left[\boldsymbol{\delta}_{1k}\ \mathrm{T}_1 \right]^T . \tag{4.31}$$

4. Updating the Weights from the INPUT to HIDDEN Layer $\mathbf{W}_{ij}$

According to the gradient algorithm, we should set

$$\Delta \mathbf{W}_{ij} \propto -\ \left[\frac{\partial E}{\partial \mathbf{W}_{ij}} \right]^T = -\left[\frac{\partial E}{\partial \boldsymbol{O}_{1j}}\ \mathrm{T} \right]^T , \tag{4.32}$$

where $\dfrac{\partial E}{\partial \boldsymbol{O}_{1j}} \in \mathbf{R}^{1 \times m}$, and $\mathrm{T} \equiv \dfrac{\partial \boldsymbol{O}_{1j}}{\partial \mathbf{W}_{ij}} \in \mathbf{R}^{m \times n \times m}$ is a three dimensional tensor since $\boldsymbol{O}_{1j} \in \mathbf{R}^{m \times 1}$ and $\mathbf{W}_{ij} \in \mathbf{R}^{m \times n}$.

To compute $\frac{\partial E}{\partial \boldsymbol{O}_{1j}}$, we have

$$\frac{\partial E}{\partial \boldsymbol{O}_{1j}} = \frac{\partial E}{\partial \boldsymbol{X}_{1j}} \frac{\partial \boldsymbol{X}_{1j}}{\partial \boldsymbol{O}_{1j}}, \tag{4.33}$$

using Eqs. (4.19) and (4.18) leads to

$$\frac{\partial \boldsymbol{X}_{1j}}{\partial \boldsymbol{O}_{1j}} = \frac{\partial\, \boldsymbol{f}_j(\boldsymbol{O}_{1j})}{\partial \boldsymbol{O}_{1j}} = diag\left[x_{1j1}(1-x_{1j1}),\ \cdots,\ x_{1jm}(1-x_{1jm})\right]. \tag{4.34}$$

Because

$$\frac{\partial E}{\partial \boldsymbol{X}_{1j}} = \sum_{k=1}^{N_2} \frac{\partial E}{\partial \boldsymbol{O}_{2k}} \frac{\partial \boldsymbol{O}_{2k}}{\partial \boldsymbol{X}_{1j}}, \tag{4.35}$$

substituting Eqs. (4.21), (4.29) into (4.35) leads to

$$\frac{\partial E}{\partial \boldsymbol{X}_{1j}} = \sum_{k=1}^{N_2} (-\,\boldsymbol{\delta}_{1k}) \frac{\partial}{\partial \boldsymbol{X}_{1j}} \left(\sum_{l=1}^{N_1} \mathbf{W}_{1lk}\ \boldsymbol{X}_{1l} \right) = \sum_{k=1}^{N_2} (-\,\boldsymbol{\delta}_{1k})\ \mathbf{W}_{1jk}. \tag{4.36}$$

Therefore, substituting Eqs. (4.34) and (4.36) into (4.33), and using the notation $\boldsymbol{\delta}_j$, we get

$$\begin{aligned} \boldsymbol{\delta}_j \ &\equiv\ -\frac{\partial E}{\partial \boldsymbol{O}_{1j}} = -\frac{\partial E}{\partial \boldsymbol{X}_{1j}} \frac{\partial \boldsymbol{X}_{1j}}{\partial \boldsymbol{O}_{1j}} \\ &=\ \left(\sum_{k=1}^{N_2} \boldsymbol{\delta}_{1k}\ \mathbf{W}_{1jk} \right)\ diag\left[x_{1j1}(1-x_{1j1}),\ \cdots,\ x_{1jm}(1-x_{1jm})\right] \in \mathbf{R}^{1\times m}. \end{aligned} \tag{4.37}$$

To compute T , from Eq. (4.19), we have the l-th component of $\boldsymbol{O}_{1j}$ as $o_{1jl} = \sum_{i=1}^{N} (\mathbf{W}_l)_{ij}\ \boldsymbol{X}_i$, $l = 1,\ 2,\ \cdots,\ m$, where $(\mathbf{W}_l)_{ij}$ is the l-th row of $\mathbf{W}_{ij}$. Then, the l-th matrix of T, $\mathbf{T}_l \equiv \frac{\partial o_{1jl}}{\partial \mathbf{W}_{ij}} \in \mathbf{R}^{n\times m}$, $l = 1,\ 2,\ \cdots,\ m$, has the form of

$$\mathbf{T}_l \ \equiv\ \frac{\partial o_{1jl}}{\partial \mathbf{W}_{ij}} = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \dfrac{\partial}{\partial (\boldsymbol{W}_l)_{ij}^T} \left(\sum_{q=1}^{N} (\boldsymbol{W}_l)_{qj}\ \boldsymbol{X}_q \right) \\ \mathbf{0} \\ \vdots \end{bmatrix}^T \tag{4.38}$$

$$= \left[\begin{array}{c} \mathbf{0} \\ \vdots \\ \dfrac{\partial}{\partial (\boldsymbol{W}_l)_{ij}^T} \left((\boldsymbol{W}_l)_{ij} \ \boldsymbol{X}_i \right) \\ \mathbf{0} \\ \vdots \end{array} \right]^T = \left[\begin{array}{c} \mathbf{0} \\ \vdots \\ (\boldsymbol{X}_i)^T \\ \mathbf{0} \\ \vdots \end{array} \right]^T \longleftarrow \text{at the } l\text{-th row.}$$

In Eq. (4.32), we set $\Delta \mathbf{W}_{ij} = -\eta \left[\dfrac{\partial E}{\partial \mathbf{W}_{ij}} \right]^T$, where $\eta > 0$ is the gain factor of the HIDDEN layer. Therefore, from Eqs. (4.37) and (4.38), we conclude

$$\Delta \mathbf{W}_{ij} = -\eta \left[\frac{\partial E}{\partial \boldsymbol{O}_{2k}} \mathrm{T} \right]^T = \eta \left[\boldsymbol{\delta}_j \mathrm{T} \right]^T . \tag{4.39}$$

5. Updating the Thresholds of the OUTPUT layer Θ_{2k}

We want to update the thresholds Θ_{2k} and Θ_{1j} by

$$\Theta_{2k}(t + \Delta t) = \Theta_{2k}(t) + \Delta \Theta_{2k} \tag{4.40}$$

$$\Theta_{1j}(t + \Delta t) = \Theta_{1j}(t) + \Delta \Theta_{1j}. \tag{4.41}$$

By the gradient algorithm, we should set

$$\Delta \Theta_{2k} \propto - \left[\frac{\partial E}{\partial \Theta_{2k}} \right]^T . \tag{4.42}$$

Using Eqs. (4.22), (4.21) and (4.20), we get

$$\begin{aligned} \frac{\partial E}{\partial \Theta_{2k}} &= \frac{\partial E}{\partial \boldsymbol{X}_{2k}} \frac{\partial \boldsymbol{X}_{2k}}{\partial \Theta_{2k}} = -\left(\boldsymbol{X}_{2k}^d - \boldsymbol{X}_{2k} \right)^T \frac{\partial}{\partial \Theta_{2k}} \left(\boldsymbol{f}_k(\boldsymbol{O}_{2k}) \right) \\ &= -\left(\boldsymbol{X}_{2k}^d - \boldsymbol{X}_{2k} \right)^T diag \left[x_{2k1}(1 - x_{2k1}), \ \cdots, \ x_{2kp}(1 - x_{2kp}) \right] . \end{aligned} \tag{4.43}$$

In Eq. (4.42), we set $\Delta \Theta_{2k} = -\eta_{1\theta} \left[\dfrac{\partial E}{\partial \Theta_{2k}} \right]^T$, where $\eta_{1\theta} > 0$ is the gain factor of the thresholds at the OUTPUT layer. Therefore, from Eqs. (4.43) and (4.29), we get

$$\Delta \Theta_{2k} = \eta_{1\theta} \ [\boldsymbol{\delta}_{1k}]^T . \tag{4.44}$$

6. Updating the Thresholds of the HIDDEN layer Θ_{1j}

Using the gradient algorithm, we should set

$$\Delta\Theta_{1j} \propto - \left[\frac{\partial E}{\partial \Theta_{1j}}\right]^T . \tag{4.45}$$

Because $\frac{\partial E}{\partial \Theta_{1j}} = \frac{\partial E}{\partial \boldsymbol{X}_{1j}} \frac{\partial \boldsymbol{X}_{1j}}{\partial \Theta_{1j}}$, using Eqs. (4.36), (4.19) and (4.18), we get

$$\begin{aligned} \frac{\partial E}{\partial \Theta_{1j}} &= - \left(\sum_{k=1}^{N_2} \boldsymbol{\delta}_{1k}\, \mathbf{W}_{1jk}\right) \frac{\partial}{\partial \Theta_{1j}} (\boldsymbol{f}_j(\boldsymbol{O}_{1j})) \qquad (4.46) \\ &= - \left(\sum_{k=1}^{N_2} \boldsymbol{\delta}_{1k}\, \mathbf{W}_{1jk}\right) \; diag\,[x_{1j1}(1-x_{1j1}),\ \cdots,\ x_{1jm}(1-x_{1jm})] . \end{aligned}$$

In Eq. (4.45), we set $\Delta\Theta_{1j} = -\,\eta_\theta \left[\frac{\partial E}{\partial \Theta_{1j}}\right]^T$, where $\eta_\theta > 0$ is the gain factor of the thresholds at the HIDDEN layer. Therefore, from Eqs. (4.46) and (4.37), we get

$$\Delta\Theta_{1j} = \eta_\theta \; [\boldsymbol{\delta}_j]^T . \tag{4.47}$$

To summarize what we have developed so far, the computation of the multi-dimensional BP algorithm is listed as follows.

1. Compute the output of the HIDDEN layer $\boldsymbol{X}_{1j}$ by Eqs. (4.19) and (4.18).
2. Compute the output of the OUTPUT layer $\boldsymbol{X}_{2k}$ by Eqs. (4.21) and (4.20).
3. Update the weights from the HIDDEN to OUTPUT layer $\mathbf{W}_{1jk}$ by Eqs. (4.23), (4.31), (4.29) and (4.30).
4. Update the weights from the INPUT to HIDDEN layer $\mathbf{W}_{ij}$ by Eqs. (4.24), (4.39), (4.37) and (4.38).
5. Update the thresholds of the OUTPUT layer Θ_{2k} by Eqs. (4.40), (4.44) and (4.29).
6. Update the thresholds of the HIDDEN layer Θ_{1j} by Eqs. (4.41), (4.47) and (4.37).

Extending the BP algorithm to a vector form shifted the complexity from the network level to the node level. Though the overall computation requirement is not reduced, it results in a set of succinct formulas and is easier to specify the I/O nodes of the NN for an MIMO mapping and to express some known coupling relations. Moreover, if the NN is implemented in software and instructions of vector operations are provided, then the programming is more efficient with this vector form of BP algorithm.

4.5 Simulation Results

To test the capability of the proposed predictor, a series of simulation experiments were conducted and the main results are summarized below. First, a two–input two–output, linear, time–invariant system

$$\begin{aligned} \dot{x}_1(t) &= x_2(t) + u_1(t) \\ \dot{x}_2(t) &= -5x_1(t) - 3x_2(t) + u_2(t) \\ y_i(t) &= 22x_i(t) + 10, \qquad i = 1,\ 2, \end{aligned} \tag{4.48}$$

is simulated with the sampling interval $T_s = 0.01$ *sec.* The inputs $u_1(t)$ and $u_2(t)$ are set as sinusoidal waves of frequency 10 Hz and magnitude 10.0. Let

$$\begin{aligned} \boldsymbol{Y}(k) &\equiv [y_1(k),\ y_2(k)]^T, \qquad \boldsymbol{U}(k) \equiv [u_1(k),\ u_2(k)]^T, \qquad \text{and} \\ \hat{\boldsymbol{Y}}(k+d/k) &\equiv [\hat{y}_1(k+d/k),\ \hat{y}_2(k+d/k)]^T. \end{aligned}$$

Then an NN–based predictor is designed with ten input nodes, five hidden nodes, and five output nodes. The inputs of the network are

$$\begin{aligned} &\boldsymbol{U}(k), \quad \boldsymbol{U}(k-1), \quad \boldsymbol{U}(k-2), \quad \boldsymbol{U}(k-3), \quad \boldsymbol{U}(k-4), \quad \text{and} \\ &\boldsymbol{Y}(k-1), \quad \boldsymbol{Y}(k-2), \quad \boldsymbol{Y}(k-3), \quad \boldsymbol{Y}(k-4), \quad \boldsymbol{Y}(k-5). \end{aligned}$$

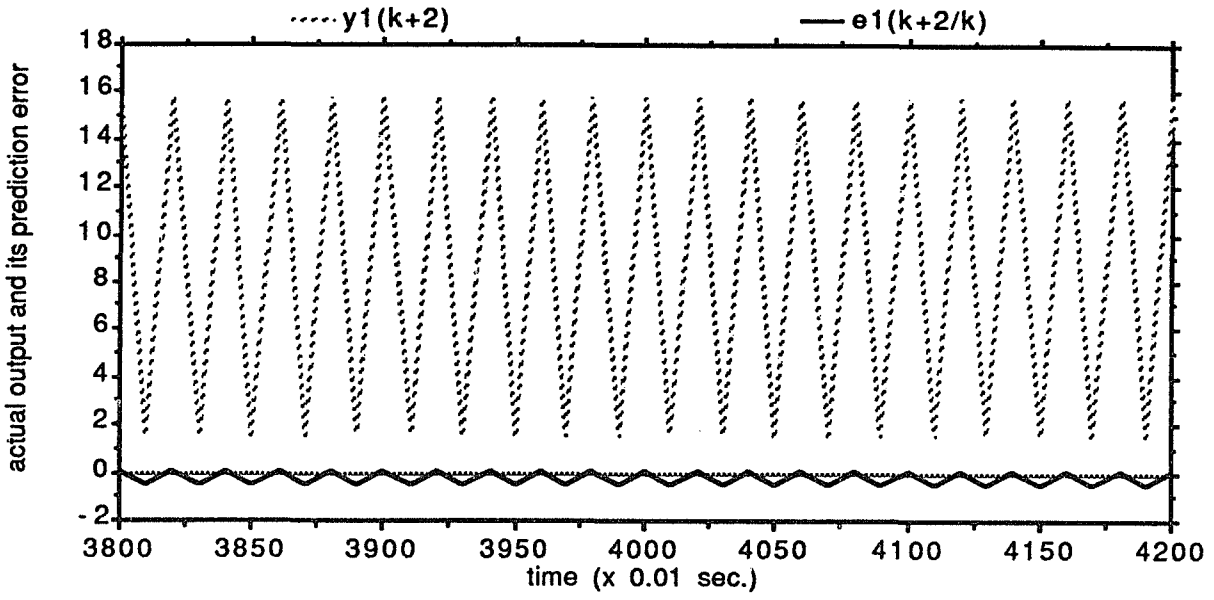


Figure 4.4: $y_1(k)$ of the nonlinear, time-invariant system and its prediction error.

The outputs are the d-step ahead predictions $\hat{\boldsymbol{Y}}(k+d/k)$, $d = 1, \cdots, 5$. The network is trained with the actual system outputs $\boldsymbol{Y}(k+d)$. The training period is 4000 sample intervals, and the relative RMS prediction errors are tabulated in Table 4.1. The network is shown to be "well trained" after 4000 sample intervals.

Using the same structure of the NN–based predictor as above, the following nonlinear, time–invariant system is tested:

$$\begin{aligned} \dot{x}_1(t) &= x_2(t) + u_1(t) \\ \dot{x}_2(t) &= -5\left(x_1^2(t) + x_1(t)\right) - 3x_2(t) + u_2(t) \\ y_i(t) &= 22x_i(t) + 10, \qquad i = 1,\ 2. \end{aligned} \tag{4.49}$$

The inputs $u_1(t)$ and $u_2(t)$ are set to be sinusoidal waves of frequency 10 Hz and magnitude 10.0. The actual values of $y_1(k)$, $y_2(k)$ and their prediction errors are plotted in Figs. 4.4 and 4.5, which have shown that the NN–based predictor works well for the nonlinear, time–invariant system.

Both Eqs. (4.48) and (4.49) are time–invariant for which the "training — op-

Sample intervals for statistics	Relative RMS prediction errors of y_1 (%)				
	prediction steps				
	1	2	3	4	5
0 – 500	20.61574	18.59294	15.80285	15.43578	17.34273
1000 – 1500	3.50542	2.99628	3.24761	4.43171	5.31734
2500 – 3000	3.08927	2.26003	2.52720	3.14259	3.43695
3500 – 4000	2.87110	1.75731	2.20443	2.60701	2.60416
4500 – 5000	2.68230	1.56914	2.17914	2.38665	2.06842
	Relative RMS prediction errors of y_2 (%)				
	prediction steps				
	1	2	3	4	5
0 – 500	17.47703	14.82588	14.74718	16.53000	18.53330
1000 – 1500	2.41279	2.68619	3.95526	5.09985	4.82806
2500 – 3000	1.86467	1.95556	2.69478	3.20278	2.90345
3500 – 4000	1.53732	1.60446	2.21500	2.35390	2.12424
4500 – 5000	1.35910	1.60634	2.09154	1.88261	1.69240

Table 4.1: The relative RMS prediction errors of the linear system.

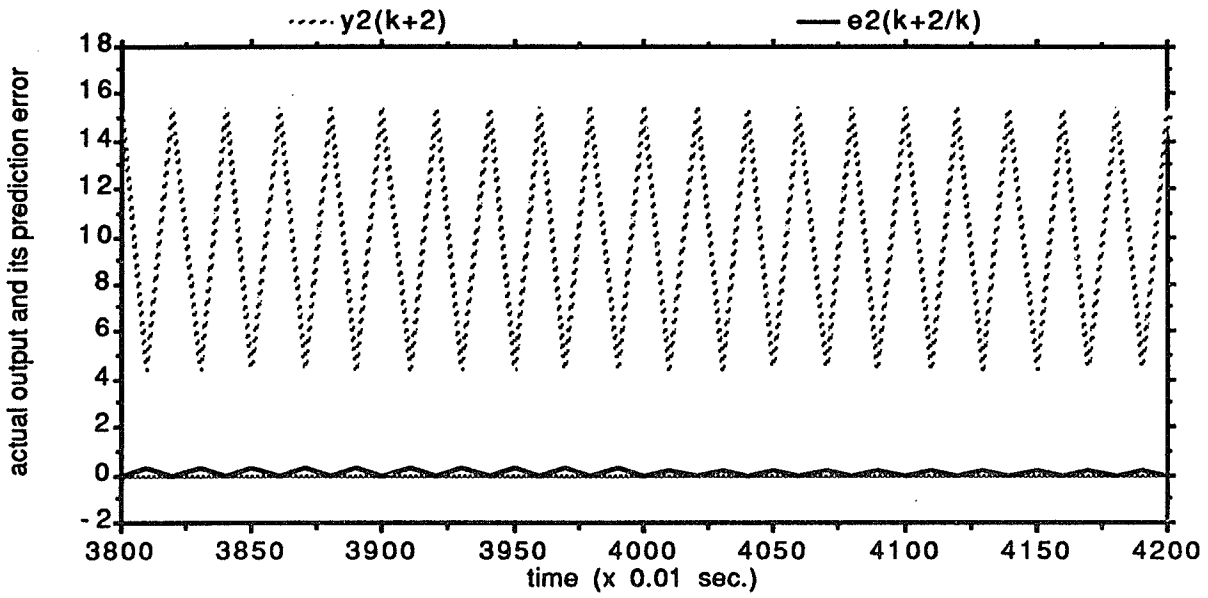


Figure 4.5: $y_2(k)$ of the nonlinear, time-invariant system and its prediction error.

eration" mode works well. However, this is not the case for time–varying systems. To test such a system, we simulated a 2–link robotic manipulator, whose dynamic equation is given as

$$\mathbf{H}(q)\ \ddot{q} + \mathbf{C}(q,\ \dot{q})\ \dot{q} + \mathbf{G}(q) = \tau$$

where $q \equiv [q_1,\ q_2]^T$ is the vector of joint positions, and $\tau \equiv [\tau_1,\ \tau_2]^T$ joint torque. The detailed explanation of this system is given in Eq. (3.19) and the appendix of this dissertation. It is a nonlinear, two–input two–output, time–varying system. The joint torque, τ, is set as sinusoidal waves of 10 Hz and magnitude 2.0 for τ_1 and 0.1 for τ_2. The sampling interval is chosen as $T_s = 0.01$ *sec.* With an initial configuration as shown in Fig. 3.6, the joint positions vary under the effects of gravitational force and joint torque. The NN–based predictor has ten input nodes, five hidden nodes, and five output nodes. The inputs of the network are

$$\tau(k), \quad \tau(k-1), \quad \tau(k-2), \quad \tau(k-3), \quad \tau(k-4), \quad \text{and}$$
$$q(k-1), \quad q(k-2), \quad q(k-3), \quad q(k-4), \quad q(k-5).$$

The outputs are the d-step ahead prediction of the joint position $\hat{\boldsymbol{q}}(k+d/k)$, $d = 1, \cdots, 5$. The simulation results are plotted in Figs. 4.6 and 4.7, for $q_1(k)$, $q_2(k)$ and their prediction errors, respectively. The training period is 4000 sampling intervals. After the training period, the prediction error increased dramatically as expected. This shows that the "training — operation" mode does not work for time–varying systems.

So, we keep track of the variation of this time–varying system by using the updating mode described in Section 4.3. The results are plotted in Figs. 4.8 and 4.9. When compared with Figs. 4.6 and 4.7, only the predictor with the updating mode is shown to work well. For this NN–based predictor, we evaluated the convergent process of the system–output prediction and the convergent process of the NN's weights. When the NN's weights are no longer changed, the NN becomes "well–trained". The simulation results have indicated that the system–output prediction converges to its true value within 200 sampling intervals. However, the NN becomes "well–trained" only after 3000 — 4000 sampling intervals. That is, the system–output prediction converges much faster than the the well–training of an NN. This again supports the idea that the NN–based predictor should be updated, but not trained. Certainly, for a time–varying system, it is meaningless to say that an NN is "well–trained".

4.6 Summary

The output prediction of a system can be represented as a mapping from its historical data and future inputs to future outputs. Even if the parameters and/or structure of the system dynamics were unknown, an NN can be designed to approximate this mapping. Using these facts, an MIMO NN–based predictor is proposed and tested for various systems. The basic structure of the predictor is determined, and

the following two major problems are solved. First, in order to track a time–varying mapping, the concept that an NN should be updated, rather than trained, is introduced and verified. By this concept and its corresponding algorithm, the proposed predictor uses only the system's historical data to adjust the weights of the NN. This also makes the network always work in a closed loop so that the reliability of the NN–based predictor is improved. Second, the BP algorithm is extended to a vector form so that an NN can be used to represent an MIMO mapping more efficiently and express some known coupling relations within the mapping more easily. This requires the nodes of the network to be capable of vector operations. Furthermore, the prediction error is analyzed and is shown to depend only on the network's error in approximating the actual mapping.

The proposed NN–based predictor has been tested for MIMO linear, nonlinear, time–invariant, and time–varying systems. All of them have shown promising results, indicating the potential use of the proposed predictor for many industrial applications.

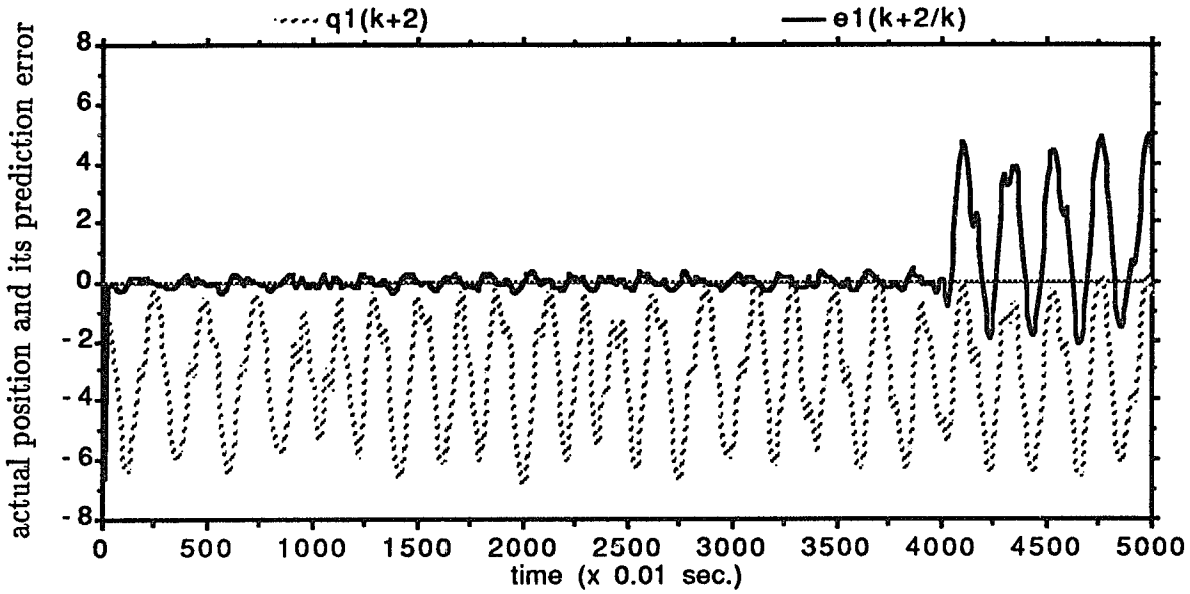


Figure 4.6: $q_1(k)$ and its prediction errors with "training — operation" mode.

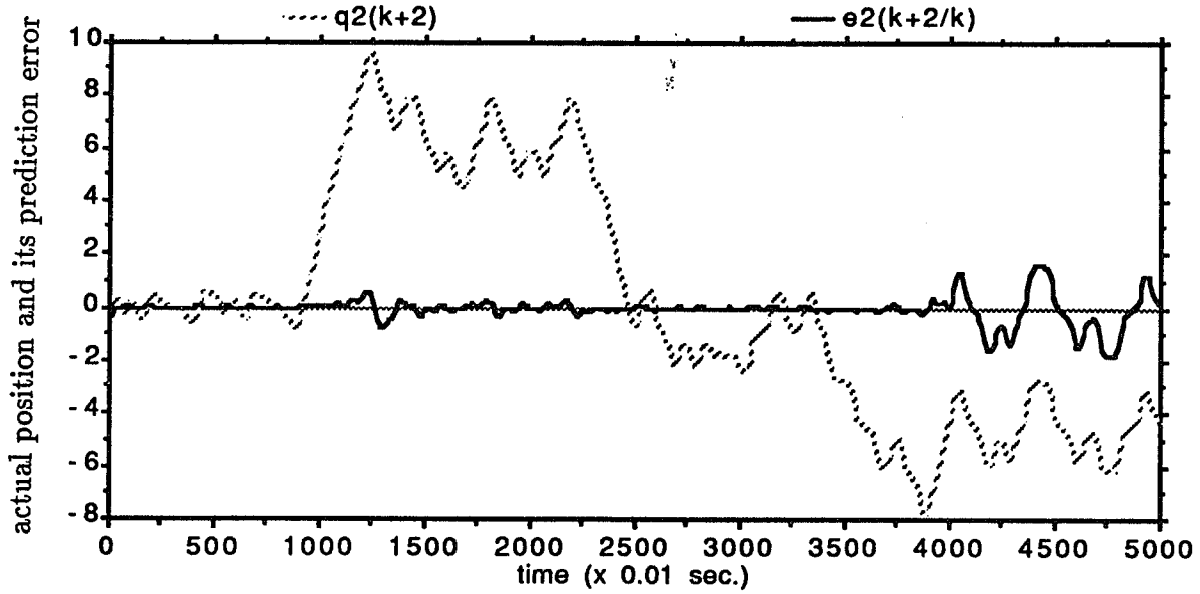


Figure 4.7: $q_2(k)$ and its prediction errors with "training — operation" mode.

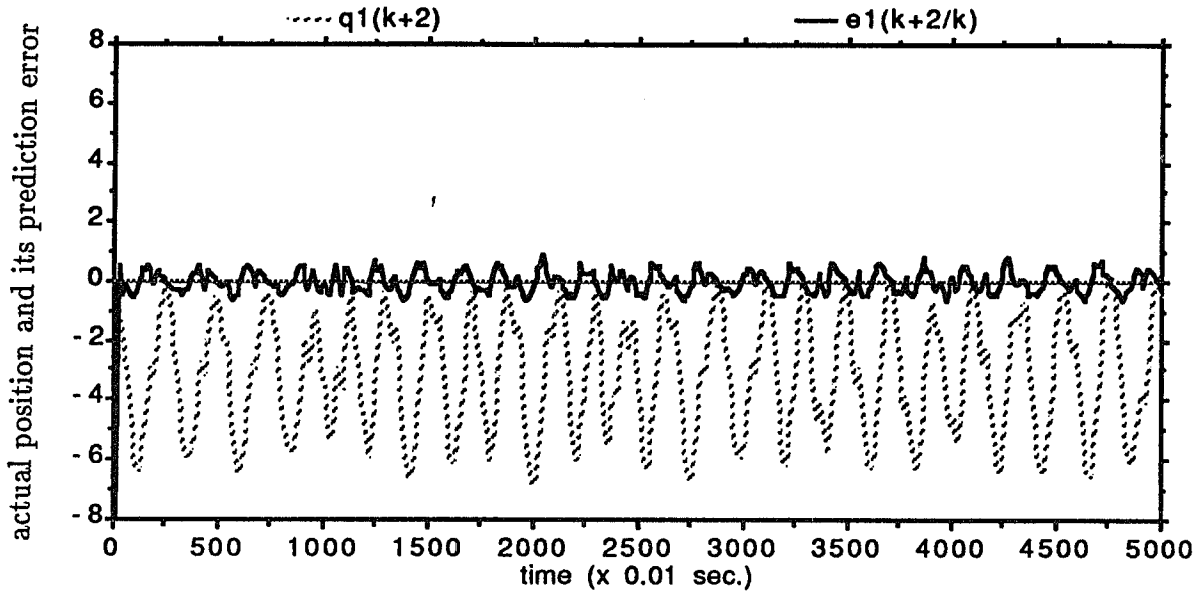


Figure 4.8: $q_1(k)$ and its prediction errors with "updating" mode.

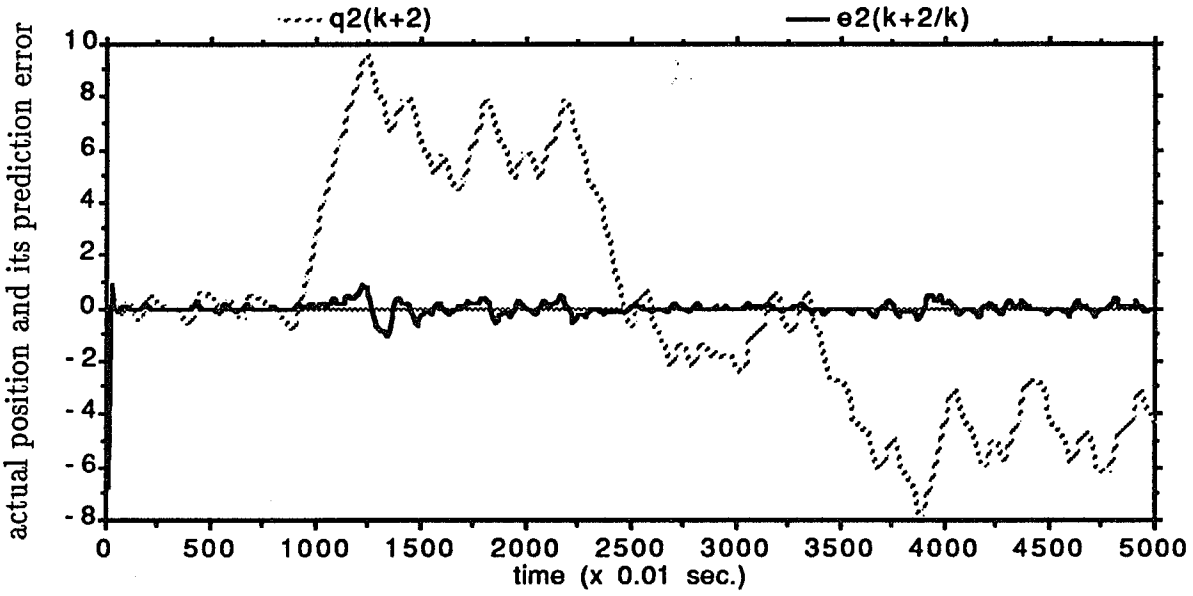


Figure 4.9: $q_2(k)$ and its prediction errors with "updating" mode.

CHAPTER V

KNOWLEDGE–BASED COORDINATOR AND ITS APPLICATION TO MULTIPLE ROBOTS

5.1 Introduction

Although some basic principles in coordinating multiple systems were developed in early 80s [LMO82], most publications addressed only conceptual interpretation, and very few dealt with actual applications. The main difficulty in coordinating multiple systems comes from the lack of precise system models and parameters as well as the lack of efficient tools for system analysis, design, and real–time computation of optimal solutions. New methods for analysis and design are thus required for the closed–loop coordination of multiple systems.

Since intelligent control does not depend only on mathematical analyses and manipulations, it is an attractive candidate for dealing with complex system control problems. An intelligent controller achieves the desired performance by searching for a goal in its knowledge base. In addition to the structures of *performance–adaptive* and *parameter–adaptive* IC introduced in Chapter 2, we have proposed a knowledge–based controller in *hierarchical structure* in Chapter 3. This knowledge–based controller is a high–level controller, which attempts to modify only the reference input to the low level. The low–level subsystem could be a servo control system, and its in-

ternal structure and parameters are not affected by adding this high–level controller. Output prediction was used to characterize system performance, and the knowledge needed to control the system was then simply represented by a decision tree.

However, all the results reported in the literature were intended for single systems. Most of the system characteristics surveyed in Section 2.2.2 may not be suitable for coordinating multiple systems, because system performance may not be easily defined and related to the measured data and control inputs. In fact, for a complex multiple–system, even a human expert's knowledge on how to coordinate it to achieve the desired performance is limited and incomplete. So, it is difficult to design a complete knowledge base for such a system. The addition of a coordinator (not necessarily an intelligent one) leads to the problem of coordinating multiple systems to form a hierarchical structure. Such an addition should not interfere in the internal structure and parameters of low–level subsystems, making the structure of performance– or parameter– adaptive intelligent controllers unsuitable for multiple–system coordination. The internal structure and/or parameters of low–level subsystems are usually not known to the coordinator. Moreover, stability analysis becomes very important, due mainly to the uncertain low–level structure and/or parameters, and to the incomplete knowledge of the coordination and system characteristics. We should, therefore, answer the following questions when designing an intelligent coordinator:

1. What are the strategy and the structure for coordinating multiple systems?
2. What are the characteristics of multiple–system performance?
3. What knowledge is necessary for coordination?
4. How should the knowledge be represented?
5. How can the qualitative knowledge be extracted from sensor data?

6. How can the result of qualitative reasoning be changed into the quantitative control signals of actuators?

7. How can system stability be analyzed and guaranteed?

A knowledge–based coordinator (KBC) for multiple systems is proposed in this chapter by combining the techniques of intelligent control and neural networks. The KBC is a high–level coordinator within a hierarchical structure. Detailed structure and/or parameters of low–level subsystems are not required by the KBC, thus allowing individual subsystems to be designed independently. This implies that some commercially–designed controllers can be coordinated to perform more sophisticated tasks than originally intended. In Section 5.2, the problem of multiple–system coordination is stated, and some basic principles of multiple–system coordination are reviewed. The proposed scheme and the assumptions are described in Section 5.3. Section 5.4 addresses the design of a KBC, including the knowledge representation, system stability, and an MIMO NN–based predictor. An example is given in Section 5.5, in which a KBC is designed and tested via simulation for coordinating two 2–link robots holding a single object. This chapter is summarized by Section 5.6.

5.2 Problem and Principles of Multiple–system Coordination

Fig. 5.1 describes two interacting systems, and this description can be easily generalized to the case of more than two systems. The system dynamics are described by

$$\mathbf{S}_1(\boldsymbol{U}_1,\ \boldsymbol{Y}_1,\ \boldsymbol{W}_2) = 0 \qquad \text{and} \qquad \mathbf{S}_2(\boldsymbol{U}_2,\ \boldsymbol{Y}_2,\ \boldsymbol{W}_1) = 0,$$

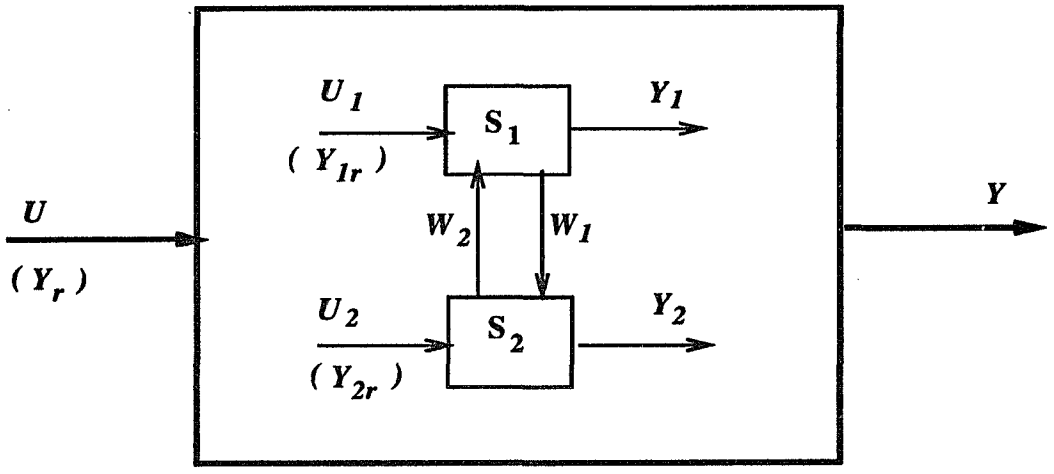


Figure 5.1: Interaction of two systems.

where $\boldsymbol{U}_i \in \mathbf{R}^{n_i}$, $\boldsymbol{W}_i \in \mathbf{R}^{m_i}$, and $\boldsymbol{Y}_i \in \mathbf{R}^{p_i}$, for $i = 1,\ 2$. Let $p = p_1 + p_2$, $n = n_1 + n_2$ and $m = m_1 + m_2$. The constraints are expressed by

$$\begin{aligned} \mathbf{S}_0 &= \{(\boldsymbol{U},\ \boldsymbol{Y},\ \boldsymbol{W}) \ :\ \mathbf{S}_1 = 0,\ \mathbf{S}_2 = 0\}, \\ \text{where}\quad \boldsymbol{U} &= \left[\boldsymbol{U}_1^T,\ \boldsymbol{U}_2^T\right]^T \in \mathbf{R}^n \text{ is the augmented control input vector,} \\ \boldsymbol{Y} &= \left[\boldsymbol{Y}_1^T,\ \boldsymbol{Y}_2^T\right]^T \in \mathbf{R}^p \text{ the augmented system output vector, and} \\ \boldsymbol{W} &= \left[\boldsymbol{W}_1^T,\ \boldsymbol{W}_2^T\right]^T \in \mathbf{R}^m \text{ the vector representing interactions between} \\ &\quad \text{the two systems.} \end{aligned}$$

Usually, the cost function of a multiple–system is the sum of the cost functions of all component systems:

$$J(\boldsymbol{U},\ \boldsymbol{Y},\ \boldsymbol{W}) \equiv J_1(\boldsymbol{U}_1,\ \boldsymbol{Y}_1,\ \boldsymbol{W}_2) + J_2(\boldsymbol{U}_2,\ \boldsymbol{Y}_2,\ \boldsymbol{W}_1). \tag{5.1}$$

The problem of coordinating multiple systems can be stated as an optimization problem: minimize the cost function, J, subject to the constraint, $\mathbf{S}_0$.

Though there are no general approaches to solving this problem for a complex multiple–system, some conceptual methods and basic principles have been suggested in [LMO82]. One of these methods is called *model coordination.* Under this method,

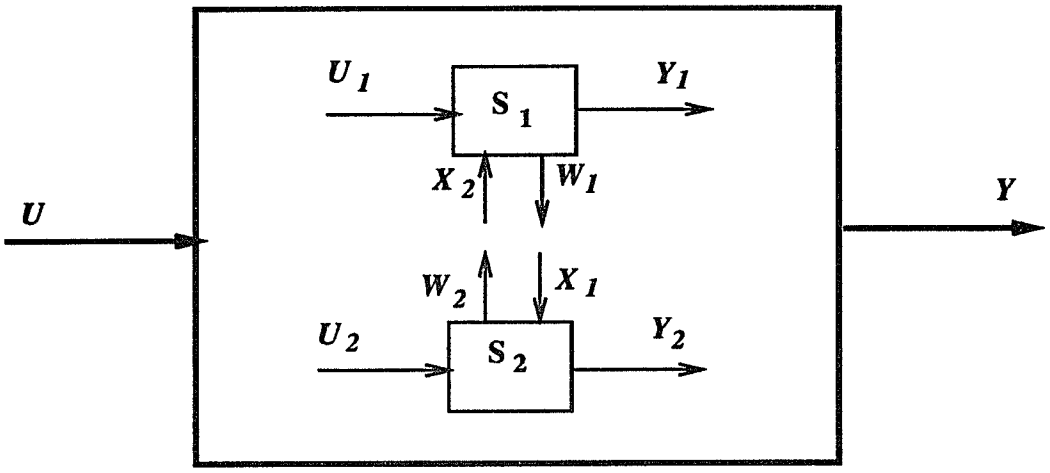


Figure 5.2: Goal Coordination of two systems.

the problem is divided into two–level optimization problems. First, suppose the interaction, $\boldsymbol{W}$, is fixed at $\boldsymbol{Z}$. Then compute

$$\mathbf{H}(\boldsymbol{Z}) = \min_{(\boldsymbol{U},\ \boldsymbol{Y},\ \boldsymbol{Z})\in \mathbf{S}_0} J(\boldsymbol{U},\ \boldsymbol{Y},\ \boldsymbol{Z}).$$

$\mathbf{H}(\boldsymbol{Z})$ is then minimized over all allowable values of $\boldsymbol{Z}$. This two–level optimization problem is solved iteratively until the desired performance is achieved. Another method is called *goal coordination* in which the system is represented as in Fig. 5.2. Suppose $\boldsymbol{W}_i$ is not necessarily equal to $\boldsymbol{X}_i$. Overall optimality is achieved by sequentially optimizing two subsystems, while treating $\boldsymbol{W}_i$ as an ordinary input variable of each corresponding subsystem. This requires $\boldsymbol{X}_i$ and $\boldsymbol{W}_i$ to be equal, which is called the *interaction balance principle.* Similar to the process of model coordination, optimality is achieved iteratively. Another basic principle of coordination, called the *interaction prediction principle,* is stated as follows. Let $\hat{\boldsymbol{W}} = \left[\hat{\boldsymbol{W}}_1^T,\ \hat{\boldsymbol{W}}_2^T\right]^T$ be the predicted interaction and $\boldsymbol{W} = \left[\boldsymbol{W}_1^T,\ \boldsymbol{W}_2^T\right]^T$ be the actual interaction under the control $\boldsymbol{U}$. Then the overall optimum will be achieved if the prediction gives the true value, that is, $\hat{\boldsymbol{W}} = \boldsymbol{W}$.

Obviously, solving these optimization problems largely depends on the knowledge

of the structure and/or dynamic parameters of low–level subsystems and mathematical synthesis. Moreover, in a hierarchical system it is desirable that adding a high–level coordinator should not affect the internal structure and/or parameters of the low–level subsystems and should only give appropriate coordination commands to them, so that each level can be designed independently of other levels. That is, the higher the level is, the more intelligence it has, and the less precise its knowledge about the low levels becomes. These requirements motivated us to design the KBC.

To design a coordinator, we first need to define a system performance index. It should be chosen to express the desired system performance and should also be amenable to some optimization methods. For example, the performance index defined in Eq. (5.1) is suitable for the concepts of model coordination and goal coordination. To design a KBC, one needs an index to explicitly express system performance; such an index will henceforth be called the *principal output.* The overall system performance index may not necessarily be the simple summation of the performance indices of all component systems. Because only system constraints are important for coordination, one may not even be able to define subsystem performance indices. Moreover, we want to relate the principal output directly to the coordination commands. The coordination commands are defined as the reference inputs to subsystems. The following sections will show that both the explicit expression for system performance and the direct relationship between the principal output and the coordination commands will simplify the design of the knowledge base and the goal–oriented search.

5.3 Description of the Principal Output Prediction Scheme

In a hierarchical structure, each level can be viewed as a mapping from its reference input to the output. The servo controller of each subsystem is usually designed separately from, and independently of, the others. In order not to interfere in the internal structure and/or parameters of the lower level, the only effective control variable is the reference input to the lower level. The reference inputs are a set of pre–designed commands, which represent the overall behavior of the multiple systems. For example, when multiple robots work in a common workspace, the reference input is the desired path and trajectory of each robot generated without considering the presence of other robots. The purpose of a high–level coordinator is to modify the desired paths and/or trajectories to avoid collision among the robots. From a high–level coordinator's point of view, the following conditions are assumed.

C1. Each subsystem is a stable, closed–loop controlled system.

C2. Each subsystem has a linear response to its reference input.

C3. Each subsystem will remain stable even during its interaction with other subsystems.

C4. System performance can be described explicitly by the principal output .

In Fig. 5.1, let $\boldsymbol{Y}$ be the principal output vector of the multiple–system, $\boldsymbol{Y}_r = [\boldsymbol{Y}_{1r}^T,\ \boldsymbol{Y}_{2r}^T]^T$ be the vector of reference input to the low level. Note now that the components of $\boldsymbol{Y}$ may not be simply the outputs of subsystems, but could be a function of these outputs:

$$\boldsymbol{Y} = \boldsymbol{F}_0(\boldsymbol{Y}_1,\ \boldsymbol{Y}_2), \qquad \text{where } \boldsymbol{F}_0 : \mathbf{R}^{p_1} \times \mathbf{R}^{p_2} \longrightarrow \mathbf{R}^{p}.$$

Because each subsystem is a closed–loop controlled system, $\boldsymbol{Y}_i$ can be represented as

$$\boldsymbol{Y}_i = \boldsymbol{f}_i(\boldsymbol{Y}_{ir},\ \boldsymbol{W}_j), \quad \text{where} \quad i,\ j = 1,\ 2, \quad j \neq i, \quad \text{and} \quad \boldsymbol{f}_i : \mathbf{R}^{n_i} \times \mathbf{R}^{m_j} \longrightarrow \mathbf{R}^{p_i}.$$

Then $\boldsymbol{Y}$ can be represented as

$$\boldsymbol{Y} = \boldsymbol{F}(\boldsymbol{Y}_{1r},\ \boldsymbol{Y}_{2r},\ \boldsymbol{W}_1,\ \boldsymbol{W}_2), \tag{5.2}$$

where $\boldsymbol{F} : \mathbf{R}^{n_1} \times \mathbf{R}^{n_2} \times \mathbf{R}^{m_1} \times \mathbf{R}^{m_2} \longrightarrow \mathbf{R}^{p}$. The principal–output vector $\boldsymbol{Y}$ in Eq. (5.2) establishes an explicit relationship between the overall system performance and the reference inputs.

Let $\hat{\boldsymbol{Y}}(k+d/k)$ and $\boldsymbol{Y}_d(k+d)$ be the d-step ahead prediction and the desired value of the principal output $\boldsymbol{Y}(k)$ at time $k+d$, respectively. Then, the performance index of the overall system can be defined as

$$J(k) = \left[\boldsymbol{Y}_d(k+d) - \hat{\boldsymbol{Y}}(k+d/k)\right]^T \left[\boldsymbol{Y}_d(k+d) - \hat{\boldsymbol{Y}}(k+d/k)\right].$$

The purpose of using a coordinator is to choose a suitable reference input vector $\boldsymbol{Y}_r(k)$ in order to minimize $J(k)$ at time k subject to a set of constraints.

Suppose the prediction of the principal output corresponding to each choice of $\boldsymbol{Y}_r(k)$ is available, and the constraints can be expressed with a set of production rules. Then, in each sampling interval, the desired performance can be obtained by iteratively trying different reference inputs and adjusting them according to the principal output prediction. For example, we propose the following algorithm to coordinate two subsystems, where the superscript i denotes the iteration count.

(1) Compute the principal output prediction $\hat{\boldsymbol{Y}}^0(k+d/k)$ for given reference inputs $\boldsymbol{Y}_{1r}^0(k)$ and $\boldsymbol{Y}_{2r}^0(k)$.

(2) Using $\hat{\boldsymbol{Y}}^i(k+d/k)$, modify the reference inputs of subsystem 1, $\boldsymbol{Y}^i_{1r}(k)$, $i = 0,\ 1,\ 2, \cdots$.

(3) Compute $\hat{\boldsymbol{Y}}^{i+1}(k+d/k)$ for given reference inputs $\boldsymbol{Y}^i_{1r}(k)$ and $\boldsymbol{Y}^0_{2r}(k)$.

(4) Set $i \leftarrow i+1$ and repeat steps (2) and (3) until $\hat{\boldsymbol{Y}}^{i+1}(k+d/k)$ cannot be improved any further with $\boldsymbol{Y}^i_{1r}(k)$ due to the constraints.

(5) Set $i \leftarrow 0$.

(6) Using $\hat{\boldsymbol{Y}}^i(k+d/k)$, modify the reference inputs of subsystem 2, $\boldsymbol{Y}^i_{2r}(k)$, $i = 0,\ 1,\ 2, \cdots$.

(7) Compute $\hat{\boldsymbol{Y}}^{i+1}(k+d/k)$ for given reference inputs $\boldsymbol{Y}^0_{1r}(k)$ and $\boldsymbol{Y}^i_{2r}(k)$.

(8) Set $i \leftarrow i+1$ and repeat steps (6) and (7) until $\hat{\boldsymbol{Y}}^{i+1}(k+d/k)$ cannot be improved any further with $\boldsymbol{Y}^i_{2r}(k)$ due to the constraints.

(9) Set $i \leftarrow 0$ and repeat steps (2) — (8) until $\hat{\boldsymbol{Y}}^i(k+d/k)$ reaches its desired value.

The conceptual structure of this scheme is given in Fig. 5.3. Obviously, this scheme needs a multiple–step predictor to compute $\hat{\boldsymbol{Y}}^i(k+d/k)$ and a KBC for the modification process of the reference inputs. By using this principal output predictor to characterize system performance, the knowledge for coordinating multiple systems becomes clear, thereby simplifying the design of a knowledge base.

We now need to address the following two problems: (1) Given the principal output prediction, how can we design this KBC? This will be discussed in the next section. (2) How can we design such a principal output predictor? In Chapter 4, an MIMO predictor has been designed using NNs. In the next sections, we emphasize to solve the first problem.

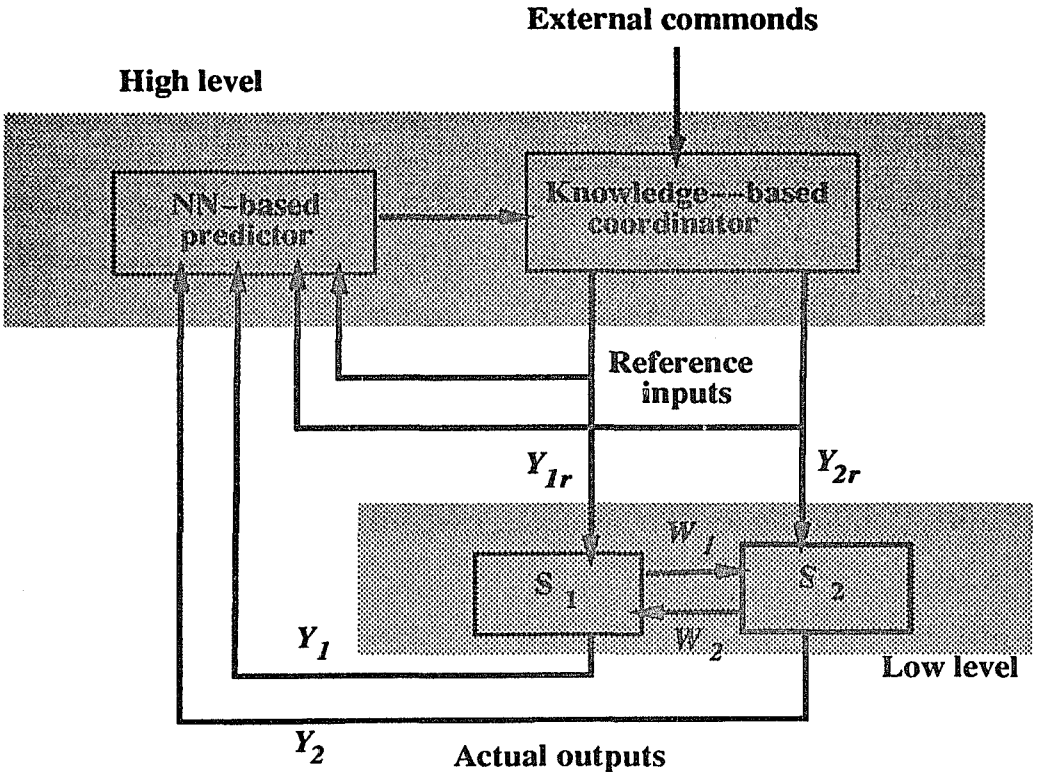


Figure 5.3: Conceptual structure of the knowledge–based coordination system.

5.4 Design of the Knowledge–Based Coordinator

A multiple–system with the KBC forms a hierarchical structure, and the low–level subsystems are viewed as a mapping from their reference input to the principal output. The goal is to modify the reference input so that the principal output reaches its desired value. For a given multiple–system we must define the principal output. Note that knowledge–based coordination is not strictly a mathematical optimization problem. The principal output must (1) have an explicit relation to the reference inputs, and (2) be measurable or computable from measured data. Because a multiple–system is designed to perform a common task(s) among the component systems, such a principal output is usually defined to express the situation of the common task(s), even though it may not explicitly reflect some of the generally–used

optimization criteria, such as energy or time.

As an example, consider the coordinated control of two robots. The two robots' operations may be tightly–coupled or loosely–coupled. They are tightly–coupled, for example, when they hold a single object rigidly and are coordinated to move the object. On the other hand, they are loosely–coupled, when they work in a common workspace and are coordinated to avoid collision. Suppose each robot is equipped with a servo controller which was originally designed for a single robot. The two robots are coordinated by modifying each robot's reference input. For the tightly–coupled case, the principal output can be defined as the object's position error or the internal/external force exerted on the object. For the loosely–coupled case, on the other hand, the positions and/or velocities of the robots' end–effectors can be used to represent the status of collision avoidance, and, thus, they are qualified to be the principal output. For both cases, an explicit relationship between system performance and reference input is established by defining the appropriate principal output.

As stated in the previous sections, we want to use the principal output predictor to see where each reference input of the subsystem will lead to. If the principal output prediction is given, simplified knowledge on how to coordinate a multiple–system can be stated in two steps:

(1) Modify the reference input and feed the modified input to the predictor.

(2) **IF** the principal output prediction yield good performance

THEN feed the reference input to the low–level subsystems

ELSE re–modify it.

Since only one reference input is modified at each time, the remaining problems

are then in which direction the reference input is modified (increase or decrease), how much it should be modified, and what are its limits? For a single–system, we have already developed such a knowledge–based controller in Chapter 3. For a multiple–system, the modification process of each reference input is similar to that of a single–system, so only the related results of Chapter 3 are summarized below.

Knowledge Representation

Using a predictor, the performance of a multiple–system is characterized by the predicted tracking error in its principal output that results from the application of the current reference input. Thus, the space of predicted tracking errors forms the input space of the KBC's knowledge base. The goal of the KBC is then to implement the modification process discussed thus far. It is not difficult to express this process in terms of a set of production rules. For each element of the reference input, the basic modification process can be represented by a decision tree as shown in Fig. 3.1. The ij-th node in the tree is represented by $([a_j^i,\ b_j^i],\ c_j^i)$, where c_j^i is the quantity added to the reference input,[1]

$$y_r^{i+1}(k) = y_r^0(k) + c_j^i.$$

$y_r^0(k)$ is an element of the original reference input vector to one of the subsystems at time k, y_r^{i+1} is its modified value after the i-th iteration, and $[a_j^i,\ b_j^i]$ is the interval to be searched, $a_j^i < c_j^i < b_j^i$ for all $i,\ j$. The process is the same as stated in Chapter 3, by giving the reference input $y_r^i(k)$, at any node $([a_j^i,\ b_j^i],\ c_j^i)$, the interval $[a_j^i,\ b_j^i]$ will be split into two subintervals $[a_k^{i+1},\ b_k^{i+1}] \equiv [a_j^i,\ c_j^i]$ and $[a_{k+1}^{i+1},\ b_{k+1}^{i+1}] \equiv [c_j^i,\ b_j^i]$, which form two successive nodes. At the i-th iteration and at the ij-th node, let the

[1]Because only the reference input to one subsystem is modified at a time, to simplify the notation, subsystem 1 and 2 will not be distinguished within this section, that is, $y_r(k)$ will represent one element of either $\boldsymbol{Y}_{1r}(k)$ or $\boldsymbol{Y}_{2r}(k)$.

predicted tracking error resulting from $y_r^i(k)$ be denoted as

$$e_j^i(k) = \hat{y}^i(k+d/k) - y_d(k+d),$$

where $y_d(k+d)$ is an element of $\boldsymbol{Y}_d(k+d)$ and $\hat{y}^i(k+d/k)$ is the corresponding element of $\hat{\boldsymbol{Y}}^i(k+d/k)$. Then, c_j^i is computed as

$$c_j^i = \begin{cases} b_j^i - (b_j^i - a_j^i)K, & \text{if} \quad e_j^i(k) < 0 \\ 0, & \text{if} \quad e_j^i(k) = 0 \\ a_j^i + (b_j^i - a_j^i)K, & \text{if} \quad e_j^i(k) > 0, \end{cases} \tag{5.3}$$

and $0 < K < 1$ is a weighting coefficient which determines the step size of the iterative operation. a_0^0 and b_0^0 are the pre–designed lower and upper bounds for the amount of reference input modification, and usually $c_0^0 = 0$.

Solution Existence and Stability Analysis

The basic forms of production rules are

IF $e_j^i(k) < 0$ **AND** $|e_j^i(k)| > \epsilon$,

THEN increase c_j^i **AND** compute $y_r^{i+1}(k) = y_r^0(k) + c_j^i$;

IF $e_j^i(k) > 0$ **AND** $|e_j^i(k)| > \epsilon$,

THEN decrease c_j^i **AND** compute $y_r^{i+1}(k) = y_r^0(k) + c_j^i$;

IF $|e_j^i(k)| \leq \epsilon$,

THEN set $y_r^{i+1}(k) = y_r^i(k)$ **AND** stop the iterative operation.

$\epsilon > 0$ is a pre–specified error tolerance. Because the amount of modification to the reference input is bounded, or $a_0^0 < c_j^i < b_0^0$ for all i, j, there may be a case where $|e_j^i(k)| > \epsilon$ for all c_j^i. This problem is solved with the same methods as stated in Chapter 3.

Suppose the prediction gives the true principal output, and let us consider the KBC and the closed-loop subsystem. The KBC can then be viewed as a map $\mathbf{M_0} : \mathbf{E} \longrightarrow \mathbf{Y}_R$, specified by all the production rules, where $\mathbf{E}$ is the space of predicted principal-output tracking error and $\mathbf{Y}_R$ the reference input space. The low-level, closed-loop subsystem is also a map, $\mathbf{L} : \mathbf{Y}_R \longrightarrow \mathbf{E}$, which is specified by the desired dynamic properties of the servo controllers. Because $\mathbf{L}$ represents a well-designed controller and there exists a reference input at time k, $\mathbf{Y}_r^i(k) \in \mathbf{Y}_R$ such that the tracking error reaches zero. Accordingly, it is reasonable to assume that $\mathbf{L}$ is a linear map. The properties of the map $\mathbf{M} \equiv \mathbf{L}\mathbf{M_0} : \mathbf{E} \longrightarrow \mathbf{E}$ depend mainly on the properties of the map $\mathbf{M_0}$. In fact, all the antecedents of production rules are based on the prediction of principal output. If the predictor gives the true principal output, then the properties of the invariant map $\mathbf{M} : \mathbf{E} \longrightarrow \mathbf{E}$ are determined solely by the knowledge base.

For system stability, all production rules in the knowledge base must form a contraction map. Similar to the arguments in Chapter 3, if (1) the principal-output prediction of a multiple-system is computable and the predictor gives the true principal output, and (2) $\mathbf{L} : \mathbf{Y}_R \longrightarrow \mathbf{E}$ of the low-level closed-loop subsystems is a linear map, and (3) the map $\mathbf{M_0} : \mathbf{E} \longrightarrow \mathbf{Y}_R$ is given by a decision tree, then we conclude that the composite map $\mathbf{M} \equiv \mathbf{L}\ \mathbf{M_0} : \mathbf{E} \longrightarrow \mathbf{E}$ is a contraction map. As pointed out in Chapter 3, at each node of the decision tree, the iterative learning process is performed and the rules always keep the search direction pointed to the node where the tracking error decreases. This implies that the iterative learning process decreases the tracking error.

Prediction of the Principal Output

Though it is assumed that the principal output, $\boldsymbol{Y}$, is measurable or computable

from the measured data, it may be very difficult to derive a closed–form expression for Eq. (5.2). To overcome this problem, we have developed an MIMO predictor using an NN in Chapter 4 in which the problem of tracking a time–varying system is solved and the BP algorithm is extended to a vector form. In what follows, only the structure of the predictor for the principal output is stated, and the detailed design procedures are the same as that described in Chapter 4.

Referring to Eq. (5.2), the d-step ahead prediction of $\boldsymbol{Y}$ can be represented by

$$\begin{aligned}
\hat{\boldsymbol{Y}}(k+d/k) &= \mathbf{F}_p(\bar{\mathbf{Y}}_{1r},\ \bar{\mathbf{Y}}_{2r},\ \bar{\mathbf{Y}}), \qquad (5.4)\\
\text{where } \bar{\mathbf{Y}}_{1r} &= (\boldsymbol{Y}_{1r}(k+i_1),\ \cdots,\ \boldsymbol{Y}_{1r}(k),\ \boldsymbol{Y}_{1r}(k-1),\ \cdots,\ \boldsymbol{Y}_{1r}(k-i_2))\\
\bar{\mathbf{Y}}_{2r} &= (\boldsymbol{Y}_{2r}(k+j_1),\ \cdots,\ \boldsymbol{Y}_{2r}(k),\ \boldsymbol{Y}_{2r}(k-1),\ \cdots,\ \boldsymbol{Y}_{2r}(k-j_2))\\
\bar{\mathbf{Y}} &= (\boldsymbol{Y}(k),\ \boldsymbol{Y}(k-1),\ \cdots,\ \boldsymbol{Y}(k-i))\\
\mathbf{F}_p :\ \mathbf{R}^{n_1} &\times \mathbf{R}^{n_2} \times \mathbf{R}^{p} \longrightarrow \mathbf{R}^{p},
\end{aligned}$$

i, i_1, i_2, j_1 and j_2 are constant integers. The interaction effects among subsystems are implicitly included in the historical data of $\bar{\mathbf{Y}}$. In Eq. (5.4), the principal output prediction is directly represented as a mapping of the reference inputs and the historical data $\bar{\mathbf{Y}}$. An NN–based predictor can be designed to learn the relationship of Eq. (5.4), using the procedures presented in Chapter 4. Fig. 5.4 shows the structure of the predictor, where the reference inputs $\bar{\mathbf{Y}}_{1r}$ and $\bar{\mathbf{Y}}_{2r}$, and the historical data $\bar{\mathbf{Y}}$ are fed to the nodes at the INPUT layer. When the NN becomes well–trained, the predictions $\hat{\boldsymbol{Y}}(k+d/k)$ for $d = 1,\ 2, \cdots$ are then produced from the OUTPUT nodes. To implement this NN–based predictor, two major problems have been solved in Chapter 4: (1) how to track a time–varying mapping, and (2) how to efficiently represent and compute an MIMO mapping with the NN. The detailed arguments are not repeated here.

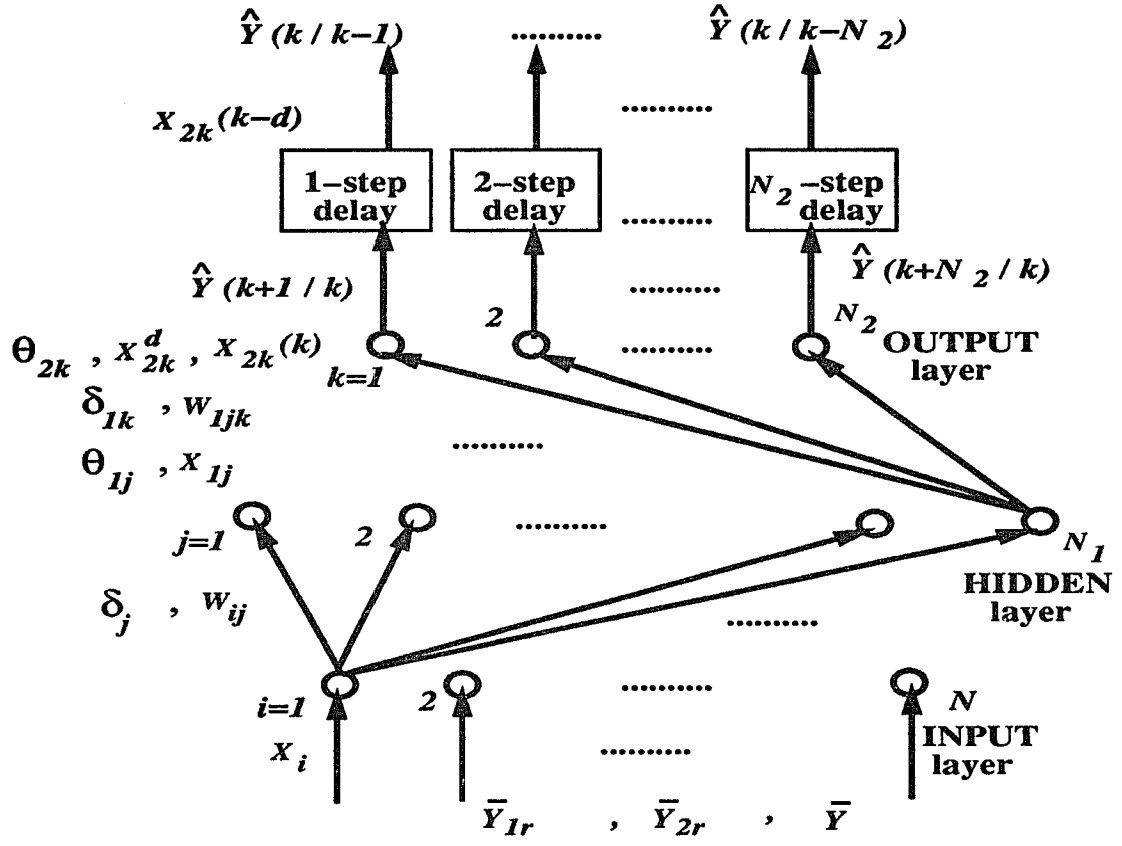


Figure 5.4: Structure of the NN–based predictor.

With the ability for learning an input–output (I/O) mapping from experience, an NN can be used to track the variation of the mapping. However, an NN alone cannot form an intelligent coordination/control system. As a general method of representing systems with learning ability, NNs lack the ability of logical reasoning and decision making, of interpreting environmental changes, and of quick response to unexpected situations. Therefore, a KBC is needed. Despite its drawbacks, the NN–based predictor establishes an explicit relationship between the principal output and the reference inputs to subsystems. Hence, the knowledge base is simplified. One can also add easily to the knowledge base such rules as the constraints of subsystems, operation monitoring, system protection, and switching of the coordination schemes. The KBC will emphasize system coordination but not data interpretation, while the

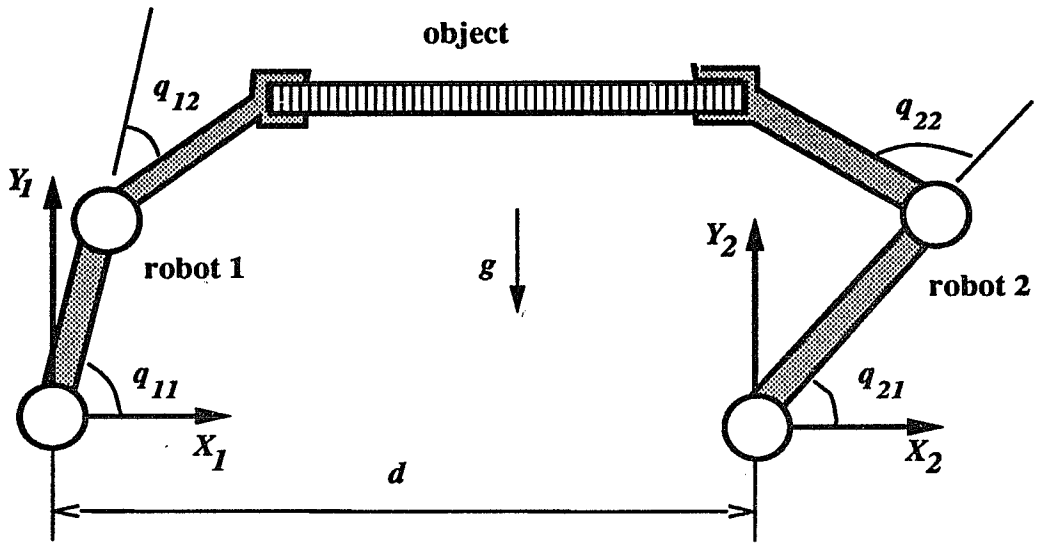


Figure 5.5: Two 2–link robots holding an object.

ability to learn will rely mainly on the NN, that is, the NN will adapt itself to the model/parameter uncertainties, disturbances, component failures, and so on.

5.5 Coordinated Control of Two 2–Link Robots

To demonstrate how to apply the proposed scheme for solving real life problems, we consider the problem of coordinating two 2–link robots holding a rigid object. The low–level subsystems include two robots each with a separately designed servo controller. The basic configuration of this example is given in Fig. 5.5. The Cartesian frame is fixed at the base of robot 1, and the trajectories of the object and the robots' end–effectors are specified relative to this frame. The task is to move the object forward and then backward in X direction while keeping the height in Y direction constant. The desired trajectory of the object are selected by a high–level planner as the reference input to the low level. If the two robots hold the object firmly, then the dynamics of the system are modeled as follows.

Dynamics of the Object

Let $\boldsymbol{f}_i = [f_{ix},\ f_{iy}]^T$ be the force exerted by the end-effector of robot i on the object in Cartesian space. Then the motion of the object is described by

$$m\ddot{\boldsymbol{P}} + m\boldsymbol{g} = \boldsymbol{f}, \qquad \boldsymbol{f} = \mathbf{W}\ \boldsymbol{F} \equiv [\mathbf{I}_2,\ \mathbf{I}_2] \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \end{bmatrix}, \tag{5.5}$$

where m is the mass of the object, $\boldsymbol{P}$ the position of the object in Cartesian space, $\boldsymbol{g}$ the gravitational acceleration, $\boldsymbol{f}$ the external force exerted on the object by the two robots, and $\mathbf{I}_2$ is a 2×2 unit matrix. From Eq. (5.5), one can see that, to achieve the object's specified acceleration, the combination of forces shared by the two robots is not unique.

Dynamics of Each Robot with Servo Controller

Suppose two robots have an identical mechanical configuration, then the force-constrained dynamic equation of robot i in joint space is given by

$$\mathbf{H}(\boldsymbol{q}_i)\ddot{\boldsymbol{q}}_i + \mathbf{C}(\boldsymbol{q}_i,\ \dot{\boldsymbol{q}}_i)\ \dot{\boldsymbol{q}}_i + \mathbf{G}(\boldsymbol{q}_i) + \mathbf{J}_i^T\ \boldsymbol{f}_i = \boldsymbol{\tau}_i, \qquad i = 1,\ 2,$$

where $\boldsymbol{q}_i = [q_{i1},\ q_{i2}]^T$ and $\boldsymbol{\tau}_i = [\tau_{i1},\ \tau_{i2}]^T$ are the vectors of the joint position and torque of robot i, respectively. $\mathbf{J}_i$ is the Jacobian matrix, and the other terms are explained in Eq. 3.19 and the appendix. Suppose both robots are position-controlled with the computed torque algorithm. That is, the control input to robot i is

$$\boldsymbol{\tau}_i = \hat{\mathbf{H}}\,(\ddot{\boldsymbol{q}}_{id} - \mathbf{K}_{Di}(\dot{\boldsymbol{q}}_i - \dot{\boldsymbol{q}}_{id}) - \mathbf{K}_{pi}(\boldsymbol{q}_i - \boldsymbol{q}_{id})) + \hat{\mathbf{h}}, \tag{5.6}$$

where $\hat{\mathbf{H}}$ and $\hat{\mathbf{h}}$ are the estimated values of $\mathbf{H}$ and $\mathbf{C}\ \dot{\boldsymbol{q}}_i + \mathbf{G}$, respectively. $\boldsymbol{q}_{id}$ is the desired value of $\boldsymbol{q}_i$, $\mathbf{K}_{Di}$ and $\mathbf{K}_{pi}$ are the controllers' gains. The reference input to the system is the desired trajectory of the object specified by $\boldsymbol{P}_d$, $\dot{\boldsymbol{P}}_d$ and $\ddot{\boldsymbol{P}}_d$, which will be transformed into the desired trajectories of the end-effector and the joints of each robot.

Problem Statement

Suppose the object is a rigid body and there is no relative motion between the end–effectors and the object. For Eq. (5.5), let $\boldsymbol{f}_d$ and $\boldsymbol{F}_d$ be the desired values of $\boldsymbol{f}$ and $\boldsymbol{F}$, respectively. Then, we have

$$\boldsymbol{F}_d = \boldsymbol{F}_{Md} + \boldsymbol{F}_{Id} \equiv \mathbf{W}^* \boldsymbol{f}_d + (\mathbf{I}_4 - \mathbf{W}^*\mathbf{W})\boldsymbol{y}_0, \tag{5.7}$$

where $\mathbf{W}^* \in \mathbf{R}^{4\times 2}$ is the pseudo–inverse of $\mathbf{W}$, $\mathbf{I}_4$ is a 4×4 unit matrix, and $\boldsymbol{y}_0 \in \mathbf{R}^4$ an arbitrary vector in the null space of $\mathbf{W}$. Therefore, the forces exerted by the end–effectors consist of two parts: $\boldsymbol{F}_{Md} = \begin{bmatrix} \boldsymbol{F}_{M1d} \\ \boldsymbol{F}_{M2d} \end{bmatrix} \in \mathbf{R}^4$ is the force to move the object and $\boldsymbol{F}_{Id} = \begin{bmatrix} \boldsymbol{F}_{I1d} \\ \boldsymbol{F}_{I2d} \end{bmatrix} \in \mathbf{R}^4$ is the internal force. The following two problems arise: (1) sharing the moving force by the two robots, and (2) changing the internal force so as to satisfy a set of constraints, such as joint torque limits or energy capacity.

In Eq. (5.7), $\boldsymbol{f}_d$ can be specified by the desired trajectory. $\boldsymbol{F}_{Id}$ is given as the desired internal force, for example, $\boldsymbol{F}_{Id} = 0$ for the least energy consumption. Because $\mathbf{W}^*$ is a constant matrix and both $\boldsymbol{f}_d$ and $\boldsymbol{F}_{Id}$ are specified, the desired force $\boldsymbol{F}_d$ is determined uniquely. However, this ideal situation of load sharing may not be achieved due to force and trajectory tracking errors. These errors may be caused by modeling/parameter errors, control performance tradeoff, and/or disturbances. It is, therefore, necessary to share the load by, or to re–assign the load to, each robot dynamically. Our goal is to design a KBC to coordinate the two robots moving the object while minimizing the internal force.

Principal Output and Its NN–Based Predictor

The reference inputs to the low–level subsystems are the desired acceleration $\ddot{\boldsymbol{P}}_{id}$, velocity $\dot{\boldsymbol{P}}_{id}$, and position $\boldsymbol{P}_{id}$ of robot i's end–effector, $i = 1,\ 2$. The internal

force can be used to evaluate system performance and has an explicit relation to the reference inputs. So, the internal force is defined as the principal output. Because the force exerted by each robot to achieve a specified acceleration of the object is not unique, it is possible to adjust the internal force by modifying the reference inputs. Since the position tracking error needs to be kept small and the desired acceleration has an explicit relationship to the force exerted on the object, only the desired acceleration is modified in order to reduce the internal force. Then, the desired acceleration issued to each robot is $\ddot{\boldsymbol{P}}_{idm}$ — the modified value of $\ddot{\boldsymbol{P}}_{id}$,

$$\ddot{\boldsymbol{P}}_{idm} = \ddot{\boldsymbol{P}}_{idc} + \ddot{\boldsymbol{P}}_{id}, \qquad i = 1,\ 2,$$

where $\ddot{\boldsymbol{P}}_{idc}$ is the increment given by the KBC to the original reference input. An NN–based predictor is designed to predict the force exerted on the object, which corresponds to each reference input. The predicted internal force (that is, the principal output) is then computed. The NN–based predictor has eight nodes at the INPUT layer, and the inputs are

$$\begin{aligned} &\boldsymbol{P}_{1d}(k), \quad \boldsymbol{P}_{1d}(k-1), \quad \boldsymbol{P}_{2d}(k), \quad \boldsymbol{P}_{2d}(k-1), \quad \text{and} \\ &\ddot{\boldsymbol{P}}_{1dm}(k), \quad \ddot{\boldsymbol{P}}_{1dm}(k-1), \quad \ddot{\boldsymbol{P}}_{2dm}(k), \quad \ddot{\boldsymbol{P}}_{2dm}(k-1). \end{aligned}$$

There are five HIDDEN nodes and six OUTPUT nodes with outputs:

$$\hat{\boldsymbol{f}}_i(k+d/k), \qquad \text{for} \qquad i = 1,\ 2, \quad d = 1,\ 2,\ 3.$$

Simulations Results

In the simulation, the task is that the two robots move the object in X direction from the initial position to the final position over one–meter distance in five seconds, and then move back to the initial position. The desired velocity and acceleration of the object are zero at both initial and final positions. The kinematic and dynamic

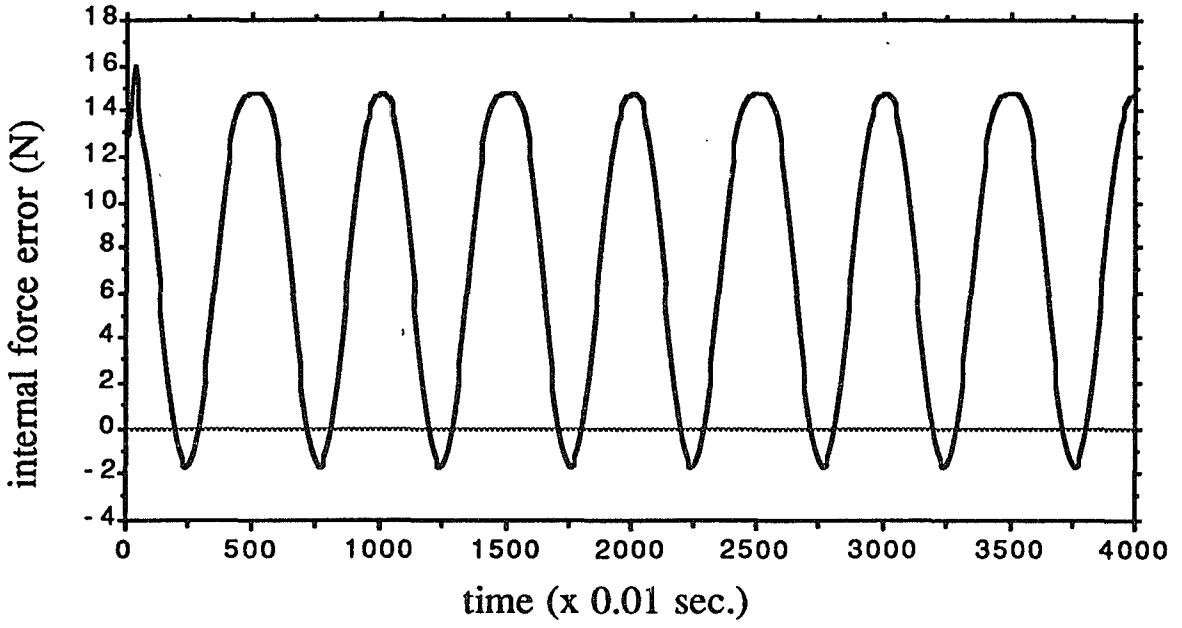


Figure 5.6: Internal force error in X direction without the KBC.

parameters of the robots are presented in Table 3.1. The sampling interval is $T_s = 0.01$ *sec.* Force predictions are used for the modification process, and position tracking is achieved by the position controllers. The 1-step ahead predictions $\hat{f}_i(k+1/k)$, $i = 1, 2$ are used in the KBC. The desired internal force is set to zero. Without the KBC, the internal force error in X direction is plotted in Fig. 5.6. After adding the KBC, the RMS error of the internal force in X direction is reduced by 63% as shown in Fig. 5.7. Moreover, both the external force error and the position tracking error are kept almost the same as those without the KBC. Detailed results are summarized in Table 5.1. Since there is no motion in Y direction, the internal force error in that direction is small enough not to require the KBC.

5.6 Summary

Focusing on the problem of coordinating multiple systems, a knowledge–based coordinator is designed using the techniques of both intelligent control and neural networks. As the high–level coordinator in a hierarchical structure, its basic principle

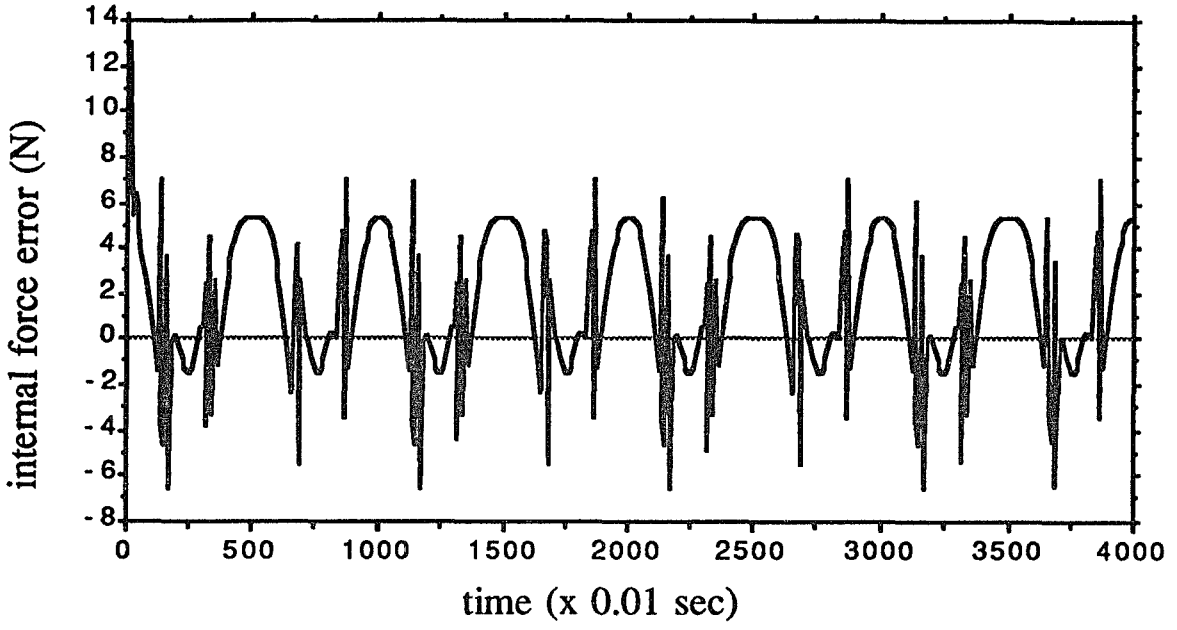


Figure 5.7: Internal force error in X direction with the KBC.

is to modify the reference inputs of low–level subsystems according to the principal output prediction in order to achieve the desired performance. By adding the proposed KBC, the internal structure and parameters of the low–level subsystems are not affected. Hence, each servo controller of the low–level subsystems can be designed separately from, and independently of, the others; no constraints need to be imposed on the design of low–level controllers. This implies that some commercially–designed servo controllers for a single system can be coordinated to work for a multiple–system.

Using the principal output and its prediction, and the structure of the decision tree for knowledge representation, the knowledge base necessary to coordinate multiple systems is greatly simplified while guaranteeing system stability. By using a predictor, the negative effects of system time delay is eliminated and each reference input is analyzed before putting it in operation. The unknown parameters and/or time–varying properties of a multiple–system are handled by the NN–based predictor, while leaving the logical reasoning and decision making on the coordination to the KBC.

To test this new scheme, the coordination problem for two 2–link robots holding a rigid object was simulated. By modifying the reference input of each robot, the internal force exerted on the object was reduced by 63%, indicating the scheme's potential for the effective coordination of multiple robots.

Sample intervals		RMS errors of internal forces (N)	
		without KBC	with KBC
0 – 1000	in X direction	9.58447	3.85020
	in Y direction	0.93141	0.53177
1001 – 2000	in X direction	9.57130	3.53340
	in Y direction	0.92339	0.49949
2001 – 3000	in X direction	9.57097	3.53688
	in Y direction	0.92339	0.49956

Sample intervals		RMS errors of external forces (N)	
		without KBC	with KBC
0 – 1000	in X direction	0.72359	0.95822
	in Y direction	2.54853	2.54345
1001 – 2000	in X direction	0.34883	0.70199
	in Y direction	0.01436	0.03345
2001 – 3000	in X direction	0.34883	0.70346
	in Y direction	0.01436	0.03355

Sample intervals		RMS tracking errors of object's positions (m)	
		without KBC	with KBC
0 – 1000	in X direction	0.03509	0.03529
	in Y direction	0.05733	0.05784
1001 – 2000	in X direction	0.03509	0.03530
	in Y direction	0.05759	0.05813
2001 – 3000	in X direction	0.03509	0.03530
	in Y direction	0.05759	0.05813

Table 5.1: The RMS errors of forces and position tracking.

CHAPTER VI

APPLICATION OF THE KBC — COLLISION AVOIDANCE IN A MULTIPLE–ROBOT SYSTEM

6.1 Introduction

Effective application of industrial robots to increase productivity and improve product quality calls for the development of an intelligent control system for them. Particularly, multiple robots need to be coordinated in order to perform such sophisticated manufacturing tasks as assembly. In a multiple–robot system, not only must each robot have good behavior, but also must multiple robots be coordinated to achieve the desired performance. One of the challenging problems in developing intelligent robot control systems is to coordinate multiple robots in a common workspace without colliding with each other.

A robot control system usually consists of a four–level hierarchy: task planning, path planning, trajectory planning, and servo control. The problem of collision avoidance among robots[1] can be solved at the path planning level by considering collision between the robot and the fixed/static obstacles in the workspace. By "path planning", we mean off–line geometric planning in robots' workspace. Generally,

[1]The term "robot" will henceforth mean "robotic manipulator" and the two terms will be used interchangeablely, unless stated otherwise.

there are two approaches to path planning: graph search and use of potential field [Kha86]. In the former, a collision–free path is obtained by searching a graph which is derived from geometric constraints. The latter assumes an artificial potential field applied to obstacles and the goal position. A collision–free path is then planned along the curve of minimum potential.

Collision avoidance can also be achieved by planning collision–free trajectories with optimization or search methods. A simple solution is to keep all other robots away from the workspace if it is occupied by a robot. Obviously, this scheme lacks the flexibility of allowing more than one robot to jointly accomplish a complex task such as assembly, and requires a longer time to complete a task, since the usage of the workspace is strictly sequential. Shin and Zheng developed a simple scheme for planning trajectories of two robots working in the same workspace by minimizing the robots' operation time while avoiding collision between them [SZ89]. Their basic idea is to delay one robot by a minimum amount of time in order to avoid collision with the other robot. However, an exact collision time between the two robots that is required for this scheme is difficult to obtain, since a precise trajectory for a given robot is not always available.

In practice, the desired path and trajectory of each robot are determined by guiding the robot through the workspace with a joystick, and its servo controller is designed independently of, and separately from, the other robots. To coordinate such robots in a common workspace, one has to devise a scheme for on–line collision detection and avoidance.

Regardless of the collision–avoidance scheme used, it is essential to track a robot's desired trajectory precisely, which in turn calls for high–performance servo controllers. Otherwise, collision may occur even if the desired trajectory is planned to

be collision–free. This implies that the dynamics of multiple robots must be figured in their coordination. An on–line coordinator is needed to guide the robots using sensory information. This on–line coordination is commonly termed the *path finding* problem. Since path finding does not always guarantee the robots to achieve their goal positions, a high–level planner is still necessary. However, the existence of on–line coordination will ease the burden on both path planning and trajectory planning for collision detection and avoidance. The path finding problem for a multiple–robot system is the main subject of this chapter.

Most industrial robots are designed to work as a stand–alone device and are usually equipped with PID–type servo controllers. Thus, it is reasonable to assume that

A1. The path of a robot is obtained by teaching, and thus, avoids collision only with fixed obstacles in the workspace.

A2. Trajectory planning does not deal with the problem of avoiding collision between moving robots.

A3. Collision avoidance is not a subject to consider when designing servo controllers. Servo controllers are commercially–designed and independent of one another.

A4. No precise knowledge of the dynamic structure and/or parameters of each robot and its servo controller is available.

There are only a few papers dealing with the on–line coordination of multiple robots for collision avoidance. The work described in [FH88] is a typical example. By representing the dynamics of each robot with a state–space equation, a nonlinear state feedback controller is designed such that the resulting closed–loop system is decoupled and linearized. The dynamics of multiple robots are also represented by

n state equations for n robots to be coordinated. A coordinator is then designed, and the essential part of the coordination command deals with the coupling effects among the links of different robots. The strategies for collision detection and avoidance are based on analytically–described avoidance trajectories, and the design procedure depends heavily on *a priori* knowledge of robots' dynamics and mathematical synthesis. The potential problems of this scheme are its computational complexity, and its restriction to the robots with cylindrical joints. Potential–field methods are also used to deal with the path finding problem [Til90, War90]. However, none of the schemes mentioned earlier satisfies all of the realistic assumptions *A1* – *A4*, because they are not intended for on–line coordination of multiple robots equipped with commercially–designed servo controllers.

It is in general very difficult to coordinate multiple robots under assumptions *A1* – *A4*. This coordination problem can be viewed as having a hierarchical structure, in which the internal structure and parameters of low–level subsystems — individual robots — must not be affected by adding a high–level coordinator. Thus, a new method must be developed to achieve the on–line coordination of multiple robots which are equipped with commercially–designed, simple controllers.

Based on a hierarchical structure, we develop a practical, yet general, design method for coordinating multiple robots in a common workspace under assumptions *A1* – *A4*. The high level consists of a knowledge–based coordinator (KBC) and a predictor. The coordinated robots form the low level. The detailed structure and/or parameters of low–level subsystems need not be known to the KBC, thus allowing the individual subsystems to be designed separately in isolation. This implies that commercially–designed robots can be coordinated to work in a common workspace without the need of modifying their servo controllers. (Such a modification is usually

very difficult to make.)

Section 6.2 states the problem of coordinating multiple robots for on–line collision avoidance and outlines our ideas in solving it. In Section 6.3, the proposed scheme is applied to coordinate two cylindrical robots for collision avoidance and is tested with extensive simulations. The scheme is also tested to coordinate two revolute robots — the second application example — in Section 6.4. In this second example, we show that collision associated with kinematic redundancy can also be handled easily by developing a set of corresponding rules for collision detection. Section 6.5 is a summary of the chapter.

6.2 Problem Statement and Basic Solution Ideas

Suppose two robots work in a common workspace. For simplicity, only the collisions in a two–dimensional workspace are considered, implying that there are no constraints on the vertical movement of the robots. Thus, for the collision avoidance problem, one can assume that each robot has three degrees of freedom (DOFs): two DOFs in $X - Y$ plane of the world coordinate, and one translational DOF in the vertical (Z) direction. In other words, the robots have three DOFs, and it is sufficient to consider collision avoidance in a two–dimensional plane. The right–of–way can be assigned to either robot whenever necessary. Then the master will follow its planned trajectory, while the slave robot must be coordinated on–line to avoid collision. The coordinator should be able to easily switch the role of the master and slave robots, when the trajectory of the slave robot cannot be modified for collision avoidance due to some physical constraints. The coordinator must not only generate a sequence of commands for collision avoidance, but also monitor each robot's dynamic response to the commands.

There are two steps for a human to avoid collision with a moving object: anticipate (estimate) the object's future position, and modify his/her original trajectory if a collision is likely to occur. This decision-making process forms the basis for our knowledge–based coordinator. We must solve the problems of (1) predicting the robots' future position, (2) developing a set of rules for collision detection based on the predicted robots' positions, and (3) modifying robots' original trajectories to avoid collision. Our solution approach is based on a hierarchical structure and satisfies all of the realistic assumptions *A1* – *A4*. In this hierarchy, the only action to take at the high level is to issue a sequence of appropriate commands to the low–level subsystems, so that the internal structure and parameters of one level will not be affected by adding another level. These commands are defined as the reference inputs to the low–level subsystems. For multiple robots in a common workspace, the reference inputs to the low level are the planned trajectory of each robot without considering the presence of other robots. The purpose of the coordinator is to modify these planned trajectories to avoid collision among the robots.

Though the master robot is supposed to follow its planned trajectory, its actual, precise position at each instant is not known due to tracking errors and system disturbances and/or noises. Hence the key problem in accomplishing on–line collision avoidance is to predict d steps ahead the robots' positions. Let $\mathbf{Y}_{id}(k)$ be the reference input of robot i, $i = 1, 2$, at time k. Suppose the robots' predicted positions corresponding to each choice of $\mathbf{Y}_{id}(k)$ are available, and other constraints can be represented by a set of production rules, such as "**IF** the trajectory of the slave robot cannot be changed due to the limits of joint torque **THEN** switch the designation of master and slave robots." In each sampling interval, collision avoidance is accomplished by iteratively trying different reference inputs and adjusting them according

to the predicted tracking error.

Our basic idea is to estimate the effects of the reference inputs with a predictor and modify them with a KBC in order to avoid collision. Using the predictor, one can foresee the effects of each reference input on the robots' future positions. Given the predicted positions of the robots, the simplified knowledge on how to coordinate the robots for collision avoidance can be stated as follows:

- Modify the reference input and feed it to the predictor.
- **IF** the predicted positions do not lead to collision **THEN** feed the reference input to the robots **ELSE** re-modify it.

By using the robots' predicted positions, the knowledge of how to coordinate multiple robots to avoid collision becomes clear, thereby simplifying the design of the KBC's knowledge base. To complete the design of the KBC, the following three problems must be addressed in detail.

(1) How is the knowledge of collision detection and avoidance acquired and represented for a specified configuration of robots?

(2) How is a multiple–step predictor designed?

(3) Given predicted robots' positions and angles, how is the KBC designed in order to modify the reference inputs of the robots?

Problem 2 and 3 have been solved in Chapter 4 and 5; therefore, this chapter will focus on the solution to problem 1.

6.3 Example 1: Coordination of Two Cylindrical Robots

The proposed scheme has a very general structure for the on–line coordination of multiple robots under realistic assumptions *A1* – *A4*. Based on robots' predicted

positions, different rules can be developed for different requirements and robots configurations, and can then be included in our knowledge–based coordinator. In this section, we will show how to derive such rules for two cylindrical robots working in a common workspace. Another example of two revolute robots will be treated in Section 6.4, exhibiting the KBC's capability in handling the collision associated with kinematic redundancy.

6.3.1 Definition of Collision

The configuration of two cylindrical robots in a common workspace is shown in Fig. 6.1. To define a collision, we use the method proposed in [FH88]. In Fig. 6.1,

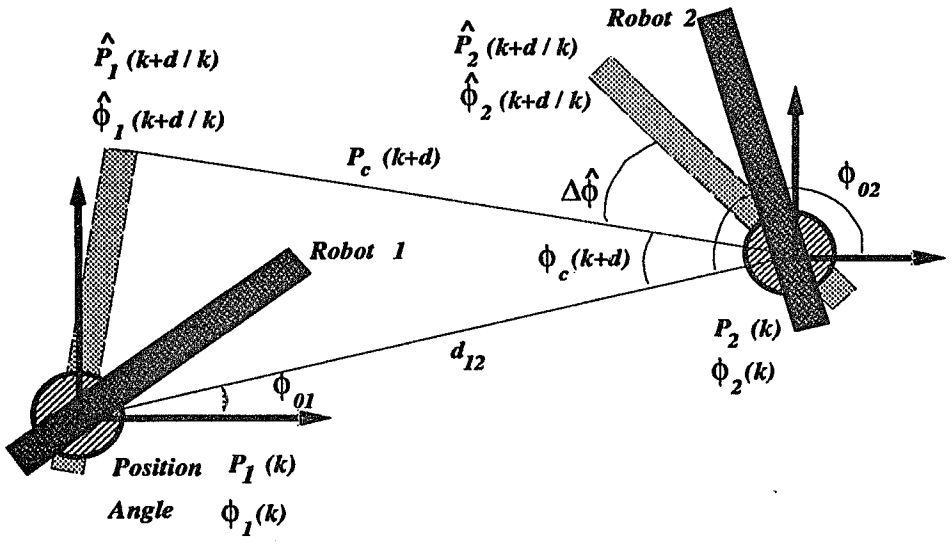


Figure 6.1: Configuration of two cylindrical robots in a common workspace.

the relationship between the base coordinates of the two robots are expressed by the distance d_{12} and the angles ϕ_{01} and ϕ_{02}. The position of robot i at time k, $P_i(k)$, is the position of its end–effector relative to its base coordinate. The angle, $\phi_i(k)$, is the angle between the robot link and the X axis of robot i's base coordinate. Their d–step ahead predictions at time k are denoted as $\hat{P}_i(k+d/k)$, $\hat{\phi}_i(k+d/k)$.

Suppose robot 1 is given the right–of–way, then a fictitious permanent colliding robot is defined with its position $P_c(k+d)$ and angle $\phi_c(k+d)$ as follows [FH88]:

$$P_c(k+d) = \sqrt{(d_{12})^2 + (\hat{P}_1(k+d/k))^2 - 2d_{12}\hat{P}_1(k+d/k)\cos(\hat{\phi}_1(k+d/k) - \phi_{01})} \tag{6.1}$$

$$\phi_c(k+d) = \arctan \frac{\hat{P}_1(k+d/k)\ \sin(\hat{\phi}_1(k+d/k))}{d_{12} - \hat{P}_1(k+d/k)\ \cos(\hat{\phi}_1(k+d/k))}. \tag{6.2}$$

To guarantee collision avoidance in the presence of tracking and prediction errors, position and angular safety margins are defined by $P_s \geq 0$ and $\phi_s \geq 0$. Without loss of generality and also for simplicity, in the following discussion we assume $\phi_{01} = 0$ and $\phi_{02} = \pi$.

6.3.2 Rules for Collision Detection

There are six different possible configurations for the two robots, and two of them — in which collision may occur — are shown in Fig. 6.2. To detect a possible collision, the estimated angular margin is defined by

$$\Delta\hat{\phi}_s = \begin{cases} \Delta\hat{\phi} - \phi_s, & \text{if } \Delta\hat{\phi} \geq 0 \\ \Delta\hat{\phi} + \phi_s, & \text{if } \Delta\hat{\phi} < 0 \end{cases} \tag{6.3}$$

$$\text{where} \quad \Delta\hat{\phi} = (\phi_{02} - \hat{\phi}_2(k+d/k)) - \phi_c(k+d).$$

Referring to Fig. 6.2, we propose the following rules for collision detection:

R3–1: **FOR** $\phi_c(k+d) \geq 0$,

IF $\Delta\hat{\phi}_s \leq 0$ **AND** $\left|\Delta\hat{\phi}_s\right| \leq |\phi_c(k+d)|$ **AND** $(\hat{P}_2(k+d/k) \geq P_c(k+d) - P_s)$,

THEN a collision is detected.

R3–2: **FOR** $\phi_c(k+d) < 0$,

IF $\Delta\hat{\phi}_s > 0$ **AND** $\left|\Delta\hat{\phi}_s\right| \leq |\phi_c(k+d)|$ **AND** $(\hat{P}_2(k+d/k) \geq P_c(k+d) - P_s)$,

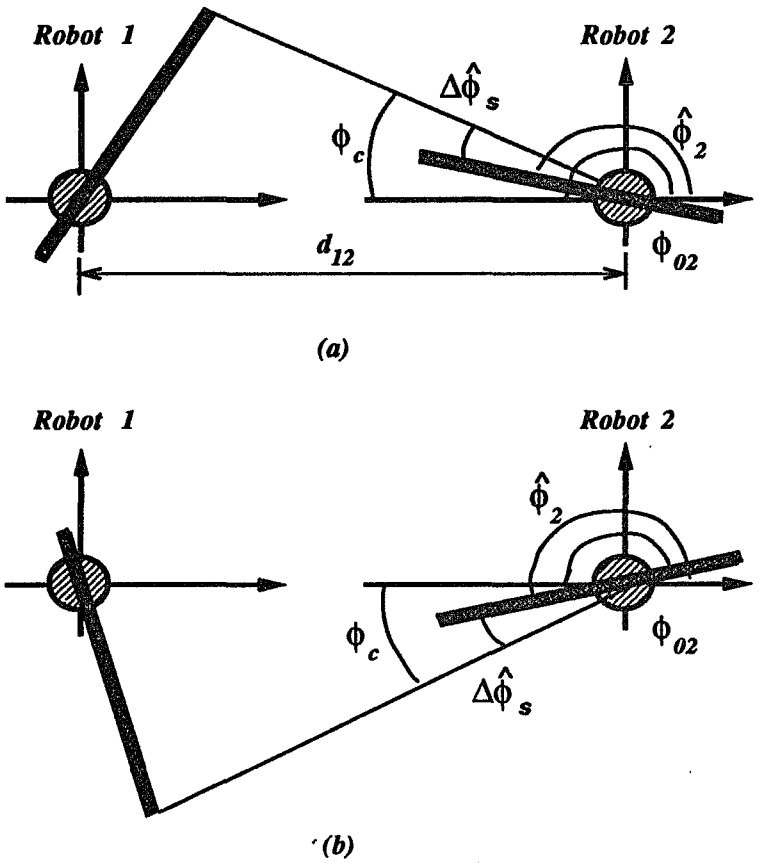


Figure 6.2: Collision detection for two cylindrical robots.

THEN a collision is detected.

Note that condition $\hat{P}_2(k+d/k) \leq P_c(k+d) - P_s$ in the above rules of collision detection is conservative. In fact, $P_c(k+d) - P_s \leq \hat{P}_2(k+d/k) < D$ may hold without causing any collision. Referring to case (a) of Fig. 6.2, D is computed as:

$$D = \begin{cases} d_{12} \left| \dfrac{\sin(\hat{\phi}_1(k+d/k))}{\sin(\hat{\phi}_1(k+d/k)+\beta)} \right| - P_s, & \text{if } \sin(\hat{\phi}_1(k+d/k)+\beta) \neq 0 \\ d_{12} - \hat{P}_1(k+d/k) - P_s, & \text{else} \end{cases} \tag{6.4}$$

where $\beta = |\phi_c(k+d)| - |\Delta\hat{\phi}_s|$. The corresponding rules of collision detection then become:

R3–3: FOR $\phi_c(k+d) \geq 0$,

IF $\Delta\hat{\phi}_s \leq 0$ **AND** $\left|\Delta\hat{\phi}_s\right| \leq |\phi_c(k+d)|$ **AND** $\hat{P}_2(k+d/k) \geq D$,

THEN a collision is detected.

R3–4: FOR $\phi_c(k+d) < 0$,

IF $\Delta\hat{\phi}_s > 0$ **AND** $\left|\Delta\hat{\phi}_s\right| \leq |\phi_c(k+d)|$ **AND** $\hat{P}_2(k+d/k) \geq D$,

THEN a collision is detected.

6.3.3 A Collision Avoidance Algorithm

We want to iteratively modify the reference inputs $\mathbf{Y}_{id}(k) \equiv [P_{id}(k),\ \phi_{id}(k)]^T$, $i = 1,\ 2$, by checking possible collisions in future. Because one of the two robots is given the right–of–way, the KBC only needs to modify the other robot's reference input for collision avoidance. The proposed algorithm for coordinating two robots to avoid collision is then given below, where the superscript i denotes the iteration count.

(1) Using the predictor, compute the predicted positions and angles of the two robots, $\hat{P}_1(k+d/k)$, $\hat{\phi}_1(k+d/k)$, $\hat{P}_2(k+d/k)$ and $\hat{\phi}_2(k+d/k)$.

(2) Compute the position and angle of the fictitious permanent colliding robot, $P_c(k+d)$ and $\phi_c(k+d)$, using Eqs. (6.1) and (6.2).

(3) Set $i \leftarrow 0$ and let $\hat{P}_2^i(k+d/k) \equiv \hat{P}_2(k+d/k)$, $\hat{\phi}_2^i(k+d/k) \equiv \hat{\phi}_2(k+d/k)$.

(4) Compute the angular margin, $\Delta\hat{\phi}_s$, using Eq. (6.3).

(5) Detect collision using R3–1 and R3–2 (or R3–3 and R3–4). If no collision is detected, then terminate.

(6) Modify the reference input of robot 2 using the KBC, and compute the corresponding predicted position and angle of robot 2, $\hat{P}_2^{i+1}(k+d/k)$ and $\hat{\phi}_2^{i+1}(k+d/k)$.

(7) Set $i \leftarrow i + 1$ and repeat steps 4—6.

The remaining problem is then to implement this algorithm in the KBC, which was treated in Chapter 5.

6.3.4 Simulation Results

The two cylindrical robots discussed so far are simulated to demonstrate the capability of the proposed scheme. The simulation is arranged as shown in Fig. 6.3, where both robots move simultaneously. Each simulated robot consists of the first

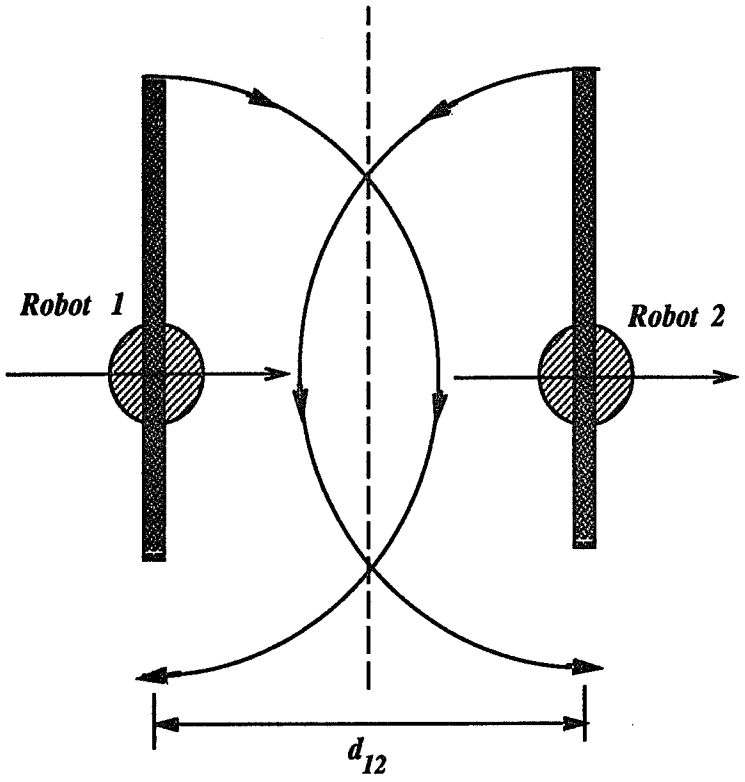


Figure 6.3: Simulation arrangement of two cylindrical robots.

and third link of a Stanford/JPL robotic manipulator [Bej74], while keeping the other joint angles at zero. The simplified dynamic model is

$$\ddot{\phi}_i = 0.7027\ \tau_{i1}$$

$$\ddot{P}_i = 0.1379\ \tau_{i3} + 8.7555, \qquad i = 1,\ 2,$$

where ϕ_i and P_i are the angle and position of robot i's joint 1 and 3, respectively, and τ_{i1}, τ_{i3} are the corresponding joint torque and force. For each robot, a PD controller is designed

$$\begin{aligned} \tau_{i1} &= 37.7(\phi_{id} - \phi_i) + 14.7(\dot{\phi}_{id} - \dot{\phi}_i), \\ \tau_{i3} &= 192.2(P_{id} - P_i) + 74.7(\dot{P}_{id} - \dot{P}_i) - 63.5. \end{aligned}$$

The safety margins are set to $P_s = 0.2\ m$ and $\phi_s = 0.25\ rad$. The sampling interval is 0.01 sec.

An NN–based predictor is designed for each robot. All the dynamic parameters and controller structure/parameters are unknown to both the NN–based predictor and the KBC. The NN–based predictor has six INPUT nodes, eight HIDDEN nodes, and six OUTPUT nodes. The inputs of the predictor are

$$P_{id}(k),\ P_{id}(k-1),\ P_{id}(k-2),\ \phi_{id}(k),\ \phi_{id}(k-1),\ \phi_{id}(k-2), \quad \text{for } i = 1, 2,$$

and the outputs of the predictor are

$$\hat{P}_i(k+1/k),\ \hat{P}_i(k+2/k),\ \hat{P}_i(k+3/k),\ \hat{\phi}_i(k+1/k),\ \hat{\phi}_i(k+2/k),\ \hat{\phi}_i(k+3/k).$$

In the KBC, 2–step ahead predictions are used to guide the modification of the reference inputs. The 2–step ahead prediction errors of robot 2 are plotted in Fig. 6.4. The NN–based predictor converges within 250 sampling intervals, though it may be far from being well trained. The results of 1–step and 3–step predictions of robot 2 as well as robot 1 are similar to the plots in Fig. 6.4.

Under the KBC's coordination, the actual paths of the two robots are plotted in Fig. 6.5. Robot 1 follows its own planned trajectory while robot 2 moves cautiously

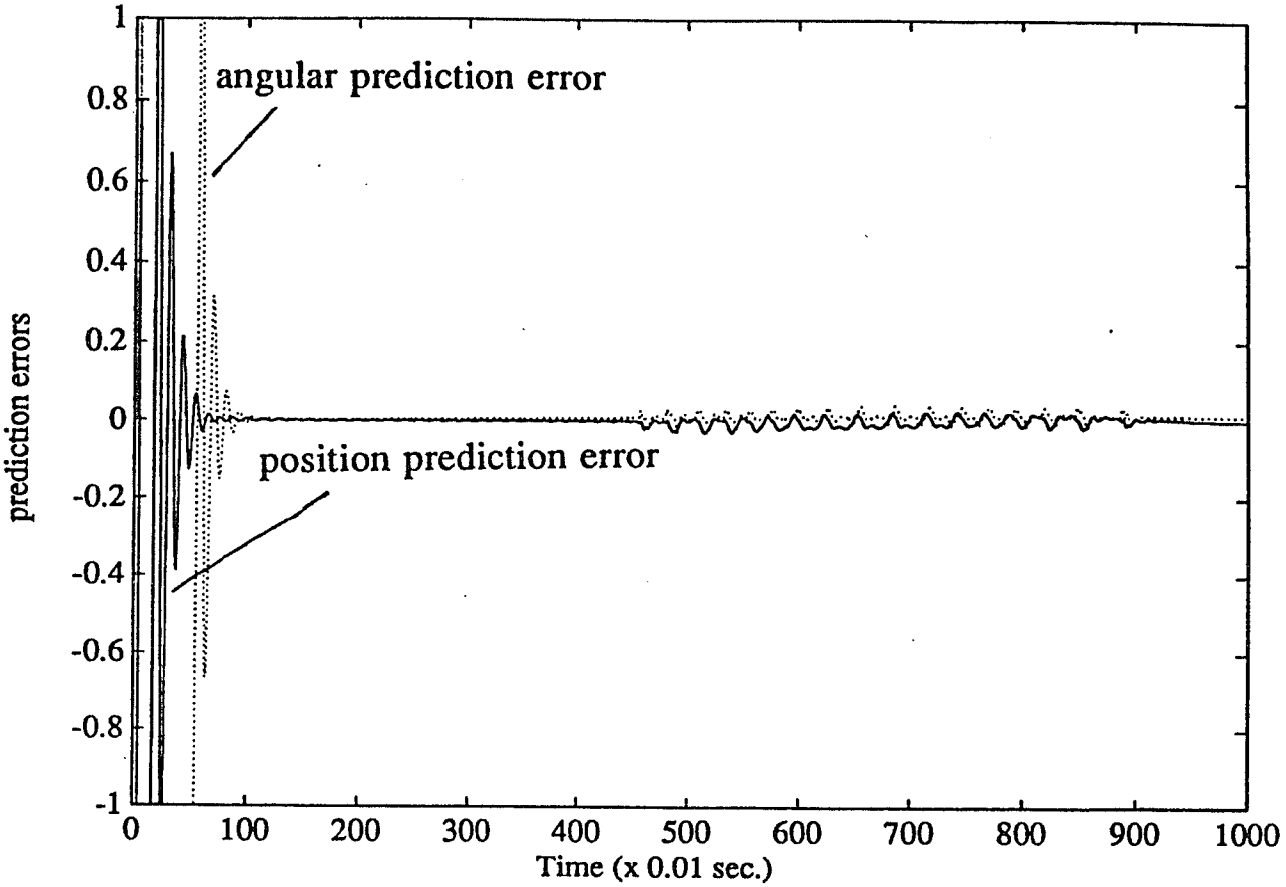


Figure 6.4: Two–step–ahead prediction errors of robot 2.

to avoid collision. In the simulation, only the position reference input of robot 2 is modified. Note that since the two robots move simultaneously, robot 2 is kept away from the working region overlapping with that of robot 1. Instead of modifying the position reference input, the angular reference input of robot 2 can be modified. In that case, robot 2 will be delayed and enter the overlapped region following robot 1. Moreover, robot 1's reference inputs can also be modified if collision cannot be avoided by modifying robot 2's reference inputs alone.

In most industrial settings, the effects of process and measurement noises must be addressed, testing not only the noise–rejection ability of each robot's servo controller, but also the capability of the KBC and the NN–based predictor. Fig. 6.6 shows the actual paths of the robots under the KBC's coordination in the presence of process noise in both robot control systems. The conditions of this simulation are the same

as above, except that the distance between the two robots $d_{12} = 1.3\ m$ and the safety margins $P_s = 0.3\ m$, $\phi_s = 0.3\ rad$. Under these conditions, the KBC still worked well in avoiding collision between the two robots.

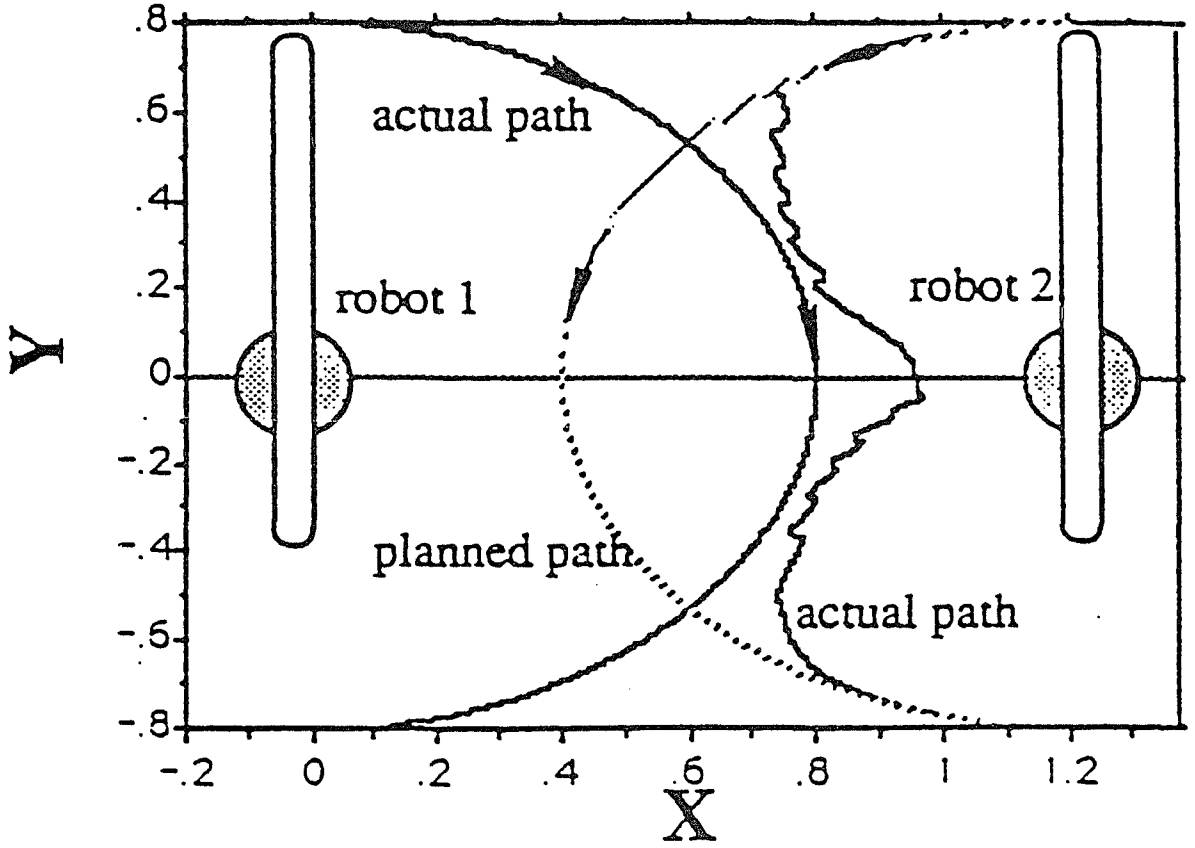


Figure 6.5: Actual paths of the two robots.

6.4 Example 2: Coordination of Two Revolute Robots

Our second example is to coordinate two revolute robots in a two–dimensional workspace, as shown in Fig. 6.7. Similar to the case of two cylindrical robots, the translational motion in Z direction is not constrained. However, in this workspace, if $\Omega_0 \subset \Omega$ is the space in which the angle of link 2 is 0 or π for each robot, then for each point $(x,\ y) \in \Omega \setminus \Omega_0$ there are two different link configurations allowing the end–effector to reach the goal point. In other words, a 2–link robot is redundant in this workspace. Robot 1 is designated as the master and will follow its planned

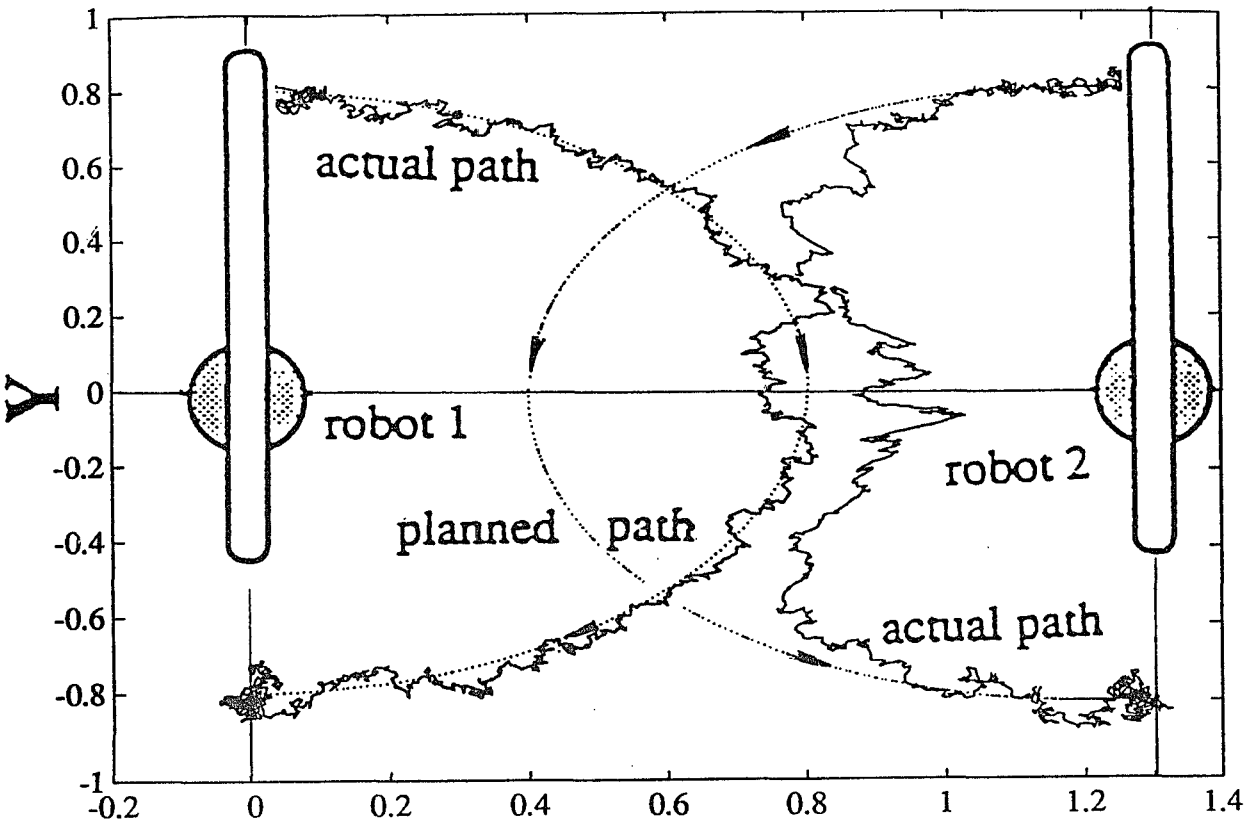


Figure 6.6: Actual paths of the two robots in the presence of process noise.

trajectory, while robot 2 is the slave and has to be coordinated for collision avoidance. Under assumptions *A1* – *A4*, this redundant case is more difficult to coordinate than cylindrical robots, especially when we have to rely solely on mathematical synthesis. However, even in such a case we can derive a set of rules for collision detection and avoidance. We want to keep the set of the rules simple, but there is a tradeoff between the simplicity of rules and the conservativeness of collision detection. In what follows, we will consider the problem of coordinating two revolute robots with emphasis on the development of rules for collision detection and strategies for collision avoidance.

6.4.1 Definition of Collisions

Consider two revolute robots working in a two–dimensional workspace, as shown in Fig. 6.7. Let $q_{i1}(k)$, $q_{i2}(k)$ be the joint angles of robot i at time k, and $\hat{q}_{i1}(k+d/k)$ and $\hat{q}_{i2}(k+d/k)$ be their d–step ahead predictions, respectively. (In what follows,

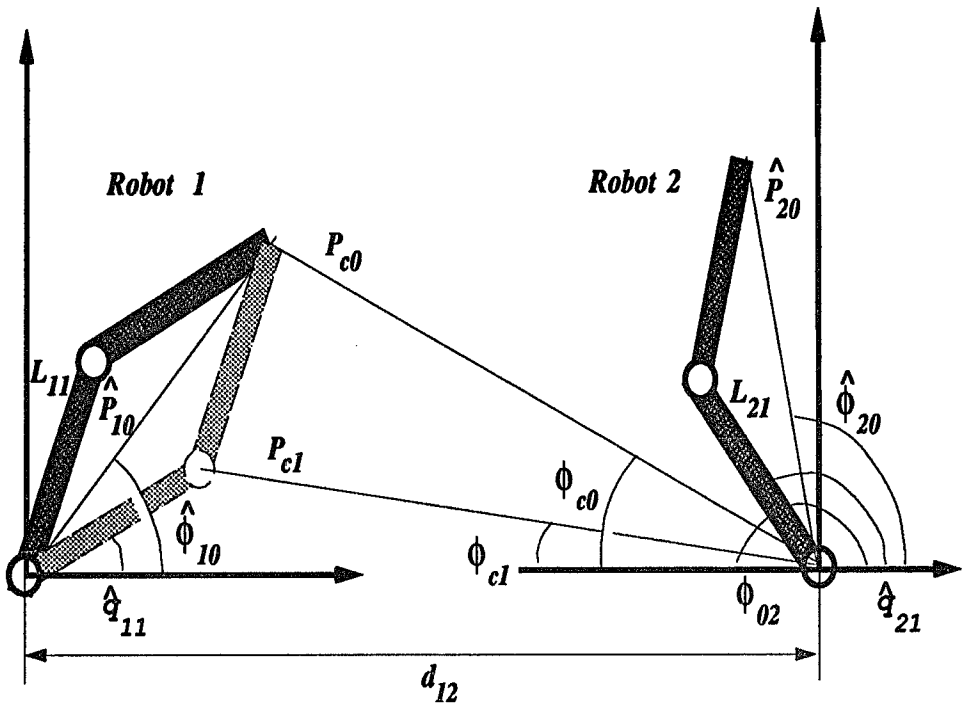


Figure 6.7: Configuration of two revolute robots in a common workspace.

$\hat{q}_{i1}(k+d/k)$ and $\hat{q}_{i2}(k+d/k)$ will be represented by $\hat{q}_{i1}$ and $\hat{q}_{i2}$, respectively, to simplify the notation.) Two fictitious permanent colliding robots — corresponding to the two possible configurations of robot 1 in Fig. 6.7 — can be defined with positions P_{c0}, P_{c1} and angles ϕ_{c0}, ϕ_{c1},

$$P_{c0} = \sqrt{(\hat{P}_{10})^2 + (d_{12})^2 - 2\ \hat{P}_{10}\ d_{12}\ \cos\hat{\phi}_{10}}, \quad \text{and} \quad \phi_{c0} = \arctan\left(\frac{\hat{P}_{10}\ \sin\hat{\phi}_{10}}{d_{12} - \cos\hat{\phi}_{10}}\right), \tag{6.5}$$

$$P_{c1} = \sqrt{(L_{11})^2 + (d_{12})^2 - 2\ L_{11}\ d_{12}\ \cos\hat{q}_{11}}, \quad \text{and} \quad \phi_{c1} = \arctan\left(\frac{L_{11}\ \sin\hat{q}_{11}}{d_{12} - \cos\hat{q}_{11}}\right), \tag{6.6}$$

where L_{ij} is the length of link j of robot i, d_{12} the distance between the bases of two robots, and

$$\begin{aligned} \hat{P}_{i0} &= \sqrt{(L_{i1})^2 + (L_{i2})^2 + 2\ L_{i1}\ L_{i2}\ \cos\hat{q}_{i2}}, \\ \hat{\phi}_{i0} &= \arcsin\left(\frac{L_{i1}\sin\hat{q}_{i1} + L_{i2}\sin(\hat{q}_{i1}+\hat{q}_{i2})}{\hat{P}_{i0}}\right), \qquad i = 1,\ 2,\ \ j = 1,\ 2. \end{aligned}$$

We want to define a permanent colliding robot corresponding to different values of P_{c0}, P_{c1} and ϕ_{c0}, ϕ_{c1}. The position of the permanent colliding robot can be conservatively selected as

$$P_c = \min(P_{c0},\ P_{c1}), \tag{6.7}$$

and its angle ϕ_c will be defined depending on different cases.

6.4.2 Rules for Collision Detection

There are three robot configurations depending on the angle of a permanent colliding robot: (1) $\phi_{c0} \geq 0$, $\phi_{c1} \geq 0$, (2) $\phi_{c0} < 0$, $\phi_{c1} < 0$, and (3) $\phi_{c0} \geq 0$, $\phi_{c1} < 0$ or $\phi_{c0} < 0$, $\phi_{c1} \geq 0$. For each of these cases, robot 2 can be represented by the predicted position $\hat{P}_2$ and the predicted angle $\hat{\phi}_2$, where

$$\hat{P}_2 = \max(\hat{P}_{20},\ L_{21}), \tag{6.8}$$

and $\hat{\phi}_2$ will be computed depending on different cases. For collision detection, the angular and length margins are defined by

$$\Delta\hat{\phi}_s = \begin{cases} \Delta\hat{\phi} - \phi_s, & \text{if } \Delta\hat{\phi} \geq 0 \\ \Delta\hat{\phi} + \phi_s, & \text{if } \Delta\hat{\phi} < 0 \end{cases} \quad \text{and} \quad \Delta\hat{P}_s = P_c - (\hat{P}_2 + P_s), \tag{6.9}$$

$$\text{where} \quad \Delta\hat{\phi} = (\phi_{02} - \hat{\phi}_2) - \phi_c.$$

Then, a set of rules for collision detection are derived for each case. Fig. 6.8 shows the collision detection of Case 1 with the safety margins $\phi_s = 0$ and $P_s = 0$. The rules for detecting collision between two revolute robots in a two–dimensional workspace are summarized below.

Case 1: Both $\phi_{c0} \geq 0$, and $\phi_{c1} \geq 0$

Define $\phi_c = \max(\phi_{c0},\ \phi_{c1})$ and $\hat{\phi}_2 = \max(\hat{\phi}_{20},\ \hat{q}_{21})$, then the rules are

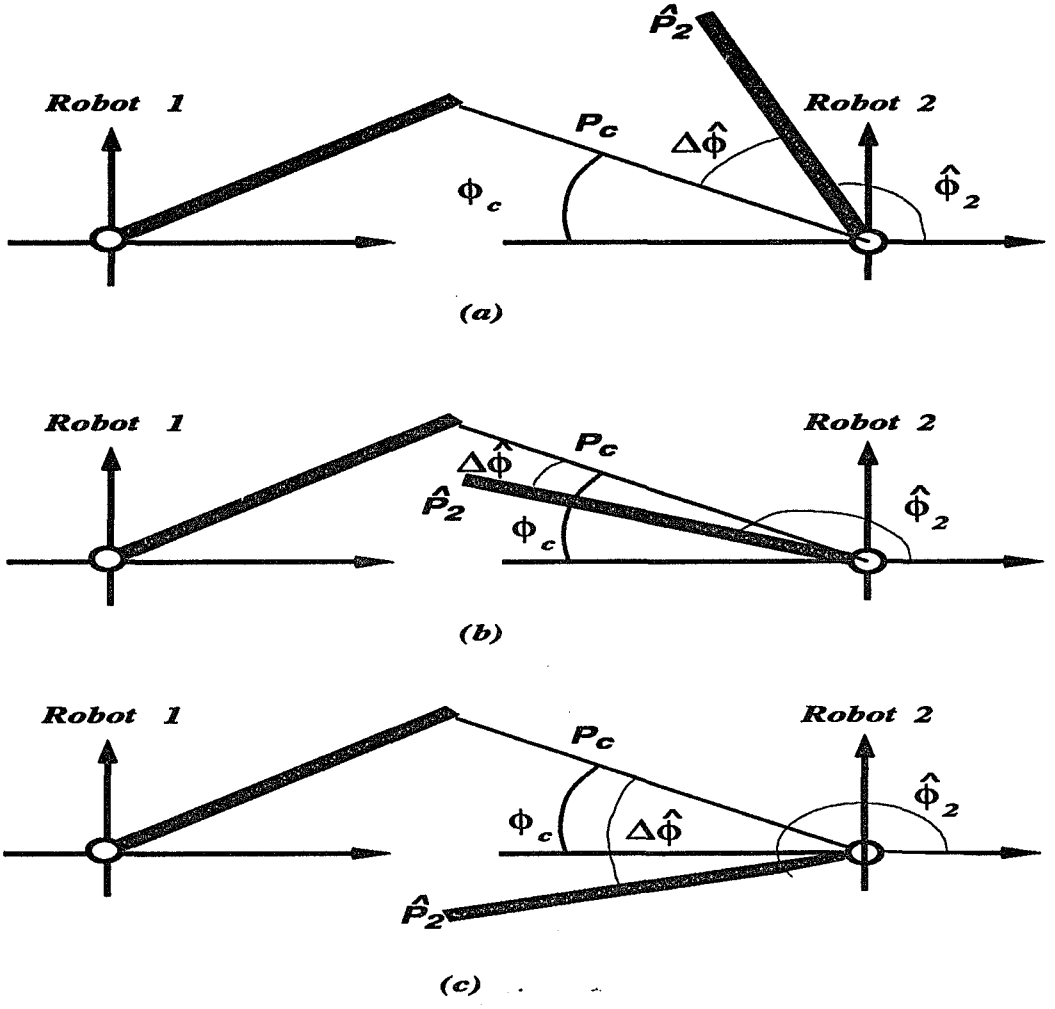


Figure 6.8: Collision detection for two revolute robots: Case 1.

R4–1: IF $\Delta\hat{\phi}_s \leq 0$ **AND** $\left|\Delta\hat{\phi}_s\right| \leq |\phi_c|$ **AND** $\Delta\hat{P}_s \leq 0$,

THEN a collision is detected.

R4–2: IF $\Delta\hat{\phi}_s \leq 0$ **AND** $\left|\Delta\hat{\phi}_s\right| > |\phi_c|$,

THEN set $\hat{\phi}_2 = \min(\hat{\phi}_{20},\ \hat{q}_{21})$, compute $\Delta\hat{\phi} = (\phi_{02} - \hat{\phi}_2) - \phi_c$ and

$\Delta\hat{\phi}_s$ as in Eq. (6.9),

IF $\left|\Delta\hat{\phi}_s\right| \leq |\phi_c|$ **AND** $\Delta\hat{P}_s \leq 0$,

THEN a collision is detected.

END

Case 2: Both $\phi_{c0} < 0$, and $\phi_{c1} < 0$

Define $\phi_c = \min(\phi_{c0},\ \phi_{c1})$ and $\hat{\phi}_2 = \min(\hat{\phi}_{20},\ \hat{q}_{21})$, then the rules are

R4–3: IF $\Delta\hat{\phi}_s \geq 0$ **AND** $\left|\Delta\hat{\phi}_s\right| \leq |\phi_c|$ **AND** $\Delta\hat{P}_s \leq 0$,

THEN a collision is detected.

R4–4: IF $\Delta\hat{\phi}_s \geq 0$ **AND** $\left|\Delta\hat{\phi}_s\right| > |\phi_c|$,

THEN set $\hat{\phi}_2 = \max(\hat{\phi}_{20},\ \hat{q}_{21})$, compute $\Delta\hat{\phi} = (\phi_{02} - \hat{\phi}_2) - \phi_c$ and

$\Delta\hat{\phi}_s$ as in Eq. (6.9),

IF $\left|\Delta\hat{\phi}_s\right| \leq |\phi_c|$ **AND** $\Delta\hat{P}_s \leq 0$,

THEN a collision is detected.

END

Case 3: $\phi_{c0} \geq 0,\ \phi_{c1} < 0$ or $\phi_{c0} < 0,\ \phi_{c1} \geq 0$.

Define $\phi_c = \max(|\phi_{c0}|,\ |\phi_{c1}|)$ and $\hat{\phi}_2 = \max(\hat{\phi}_{20},\ \hat{q}_{21})$, then the rules are

R4–5: IF $\Delta\hat{\phi}_s \leq 0$ **AND** $\left|\Delta\hat{\phi}_s\right| \leq |2\ \phi_c|$ **AND** $\Delta\hat{P}_s \leq 0$,

THEN a collision is detected.

R4–6: IF $\Delta\hat{\phi}_s \leq 0$ **AND** $\left|\Delta\hat{\phi}_s\right| > |2\ \phi_c|$,

THEN set $\hat{\phi}_2 = \min(\hat{\phi}_{20},\ \hat{q}_{21})$, compute $\Delta\hat{\phi} = (\phi_{02} - \hat{\phi}_2) - \phi_c$ and

$\Delta\hat{\phi}_s$ as in Eq. (6.9),

IF $\left|\Delta\hat{\phi}_s\right| \leq |2\ \phi_c|$ **AND** $\Delta\hat{P}_s \leq 0$,

THEN a collision is detected.

END

Clearly, the rules for collision detection are much more complicated than those for two cylindrical robots because of the kinematic redundancy. However, these rules can be simplified if the planned trajectories of the two robots are given *a priori*, as discussed in Section 6.4.4.

6.4.3 Strategies for Collision Avoidance

It is assumed that the desired trajectories of both robots are planned while considering fixed obstacles (not the moving robots) in the workspace. Using a multi–step predictor and the rules developed above, a possible collision can be detected. Once a collision is detected, the planned trajectory of robot 2 will be modified using the algorithm in Section 6.3.3 in order to avoid the collision. Though either increasing or decreasing the speed of robot 2 may avoid the collision, the reasonable maneuver is to slow down robot 2 since the maximum speed and acceleration/deceleration are usually bounded. This implies that the reference input be modified in one direction (that is, decrease). In order to give robot 2 a sufficient time so that it can maneuver to avoid the anticipated collision, the safety margins ϕ_s and P_s are added to the length and angular margins as in Eq. (6.9). Moreover, it is possible to modify the planned trajectory of one joint to avoid collision, which will in turn simplify the modification process of the reference inputs.

6.4.4 Simulation Results

The robots are arranged as shown in Fig. 6.9, and the dynamic and kinematic parameters of the two robots are identical and shown in Table 3.1. It is assumed that each robot is equipped with a PD–type servo controller, and the sampling interval is 0.01 sec. The planned trajectories are symmetric, and their values at some special points are given in Table 6.1, where the subscript d denotes the desired value.

An NN–based predictor is designed for each robot. There are six INPUT nodes with inputs

$$q_{i1d}(k),\ q_{i1d}(k-1),\ q_{i1d}(k-2),\ q_{i2d}(k),\ q_{i2d}(k-1),\ q_{i2d}(k-2),$$

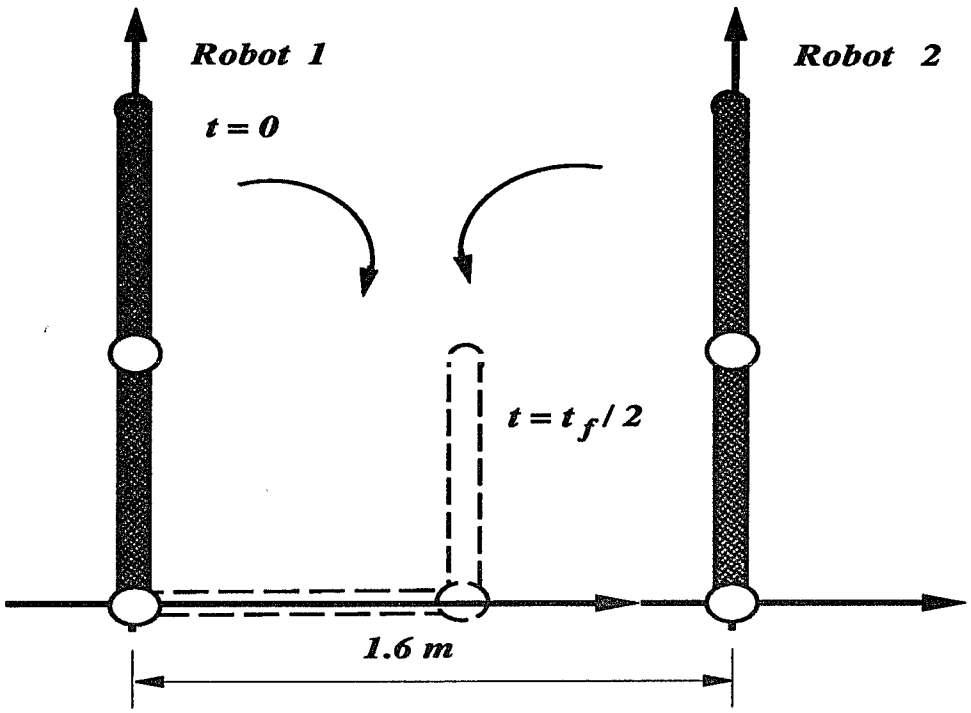


Figure 6.9: Simulation arrangement of two revolute robots.

eight HIDDEN nodes and six OUTPUT nodes with outputs,

$$\hat{q}_{i1}(k+1/k),\ \hat{q}_{i1}(k+2/k),\ \hat{q}_{i1}(k+3/k),\ \hat{q}_{i2}(k+1/k),\ \hat{q}_{i2}(k+2/k),\ \hat{q}_{i2}(k+3/k),$$

for robot i, $i = 1,\ 2$.

Considering the specific configuration and the planned trajectories of this example, the rules for collision detection can be simplified. Since the moving directions of both robots are the same in this example, one can always derive a suitable permanent colliding robot by choosing

$$\phi_c = \max(\phi_{c0},\ \phi_{c1}), \quad \text{and} \quad P_c = \min(P_{c0},\ P_{c1}).$$

The rules for collision detection can then be simplified and are listed below.

(1) Compute $\hat{\phi}_2 = \max(\hat{\phi}_{20},\ \hat{q}_{21})$, and $\hat{P}_2 = \max(\hat{P}_{20},\ L_{21})$.

(2) Compute angular and length margins $\Delta\hat{\phi}_s$, and $\Delta\hat{P}_s$ as in Eq. (6.9).

Robot 1	Joint 1			Joint 2		
	q_{11d}	$\dot{q}_{11d}$	$\ddot{q}_{11d}$	q_{12d}	$\dot{q}_{12d}$	$\ddot{q}_{12d}$
initial values, $t = t_0$	90°	0	0	0°	0	0
middle point, $t = t_f/2$	0°	max	max	90°	0	0
final values, $t = t_f$	−90°	0	0	0°	0	0
Robot 2	**Joint 1**			**Joint 2**		
	q_{21d}	$\dot{q}_{21d}$	$\ddot{q}_{21d}$	q_{22d}	$\dot{q}_{22d}$	$\ddot{q}_{22d}$
initial values, $t = t_0$	90°	0	0	0°	0	0
middle point, $t = t_f/2$	180°	max	max	−90°	0	0
final values, $t = t_f$	270°	0	0	0°	0	0

Table 6.1: Trajectory specification of the two revolute robots.

(3) Detect collision using the following rules:

R4–7: IF $\phi_c \geq 0$ **AND** $\Delta\hat{\phi}_s \leq 0$ **AND** $\left|\Delta\hat{\phi}_s\right| \leq |\phi_c|$ **AND** $\Delta\hat{P}_s \leq 0$,

THEN a collision is detected.

R4–8: IF $\phi_c < 0$ **AND** $\Delta\hat{\phi}_s \geq 0$ **AND** $\left|\Delta\hat{\phi}_s\right| \leq |\phi_c|$ **AND** $\Delta\hat{P}_s \leq 0$,

THEN a collision is detected.

The angular and length safety margins are set to $\phi_s = 0.2\ rad.$ and $P_s = 0.05\ m$, and 2–step ahead predictions are used for collision detection. Fig. 6.10 shows the actual trajectory of robot 2, showing the slow–down of robot 2 to avoid collision. The detailed process of collision detection and avoidance is shown in Fig. 6.11 in which the collision region is divided into four sub-regions. Using rule R4–7, in region 1 collision is avoided by keeping $\Delta\hat{\phi}_s > 0$. In regions 2 – 4, we have $\left|\Delta\hat{\phi}_s\right| > |\phi_c|$,

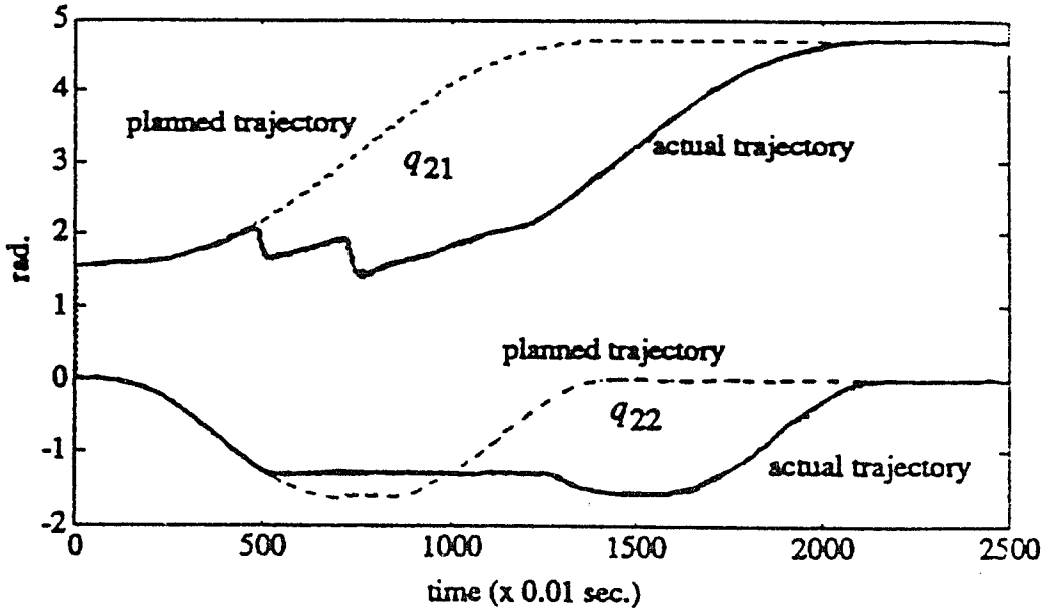


Figure 6.10: The actual trajectories of robot 2.

$\Delta\hat{P}_s > 0$, and $\Delta\hat{\phi}_s < 0$, respectively. That is, rule R4–8 is never met, so no collision occurs. These results indicate that the robots can be successfully coordinated by the KBC to avoid collision.

6.5 Summary

A new knowledge–based, hierarchical scheme is proposed to coordinate multiple robots in a common workspace for on–line detection and avoidance of collision among them. The proposed KBC and NN–based predictor form the high level of this hierarchy, and the robots to be coordinated form the low level. The KBC foresees the effects of each reference input on the low level by using the predictor, and modifies the reference input based on the prediction results in order to avoid collision among the robots.

The proposed scheme assumed that both path planning and trajectory planning did not consider on–line collision detection and avoidance, and adding the KBC did

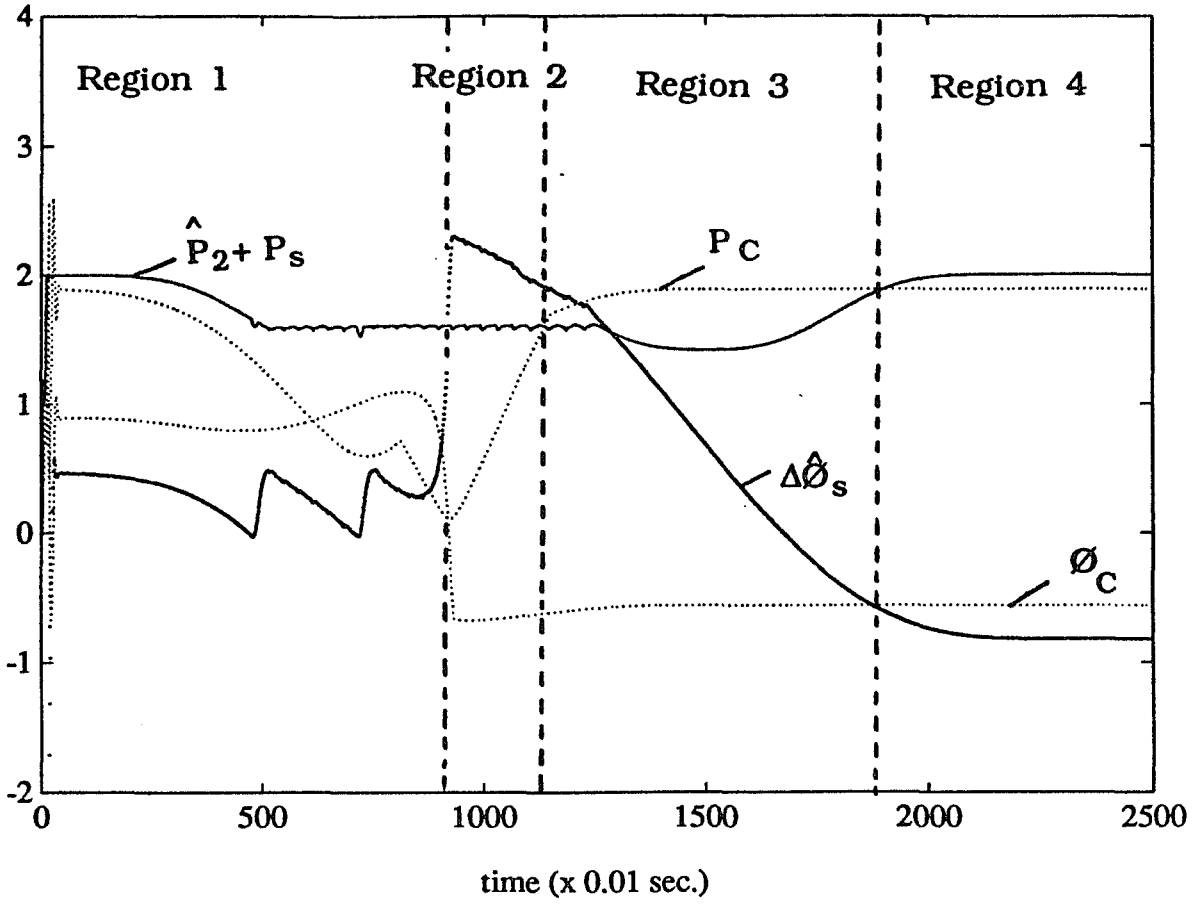


Figure 6.11: Collision detection and avoidance with the KBC.

not impose any constraints on the design of robots' servo controllers. This may relax the usual requirements (for example, the knowledge of exact dynamics) imposed on path planning, trajectory planning, and servo controllers design. Other constraints, such as joint torque limits and master/slave assignment, can be easily added to the knowledge base. Since the internal structure and parameters of the individual robot's control system are not affected, one can coordinate multiple robots — equipped with built–in servo controllers — working in a common workspace.

Though both the examples presented in this paper are concerned with two robots, it is not difficult to extend the results to the case of more than two robots, because different rules of collision detection and avoidance can be added in for different robots' configurations. Once a possible collision is detected, it can be avoided by iteratively modifying the reference input of each robot using the KBC. The simple structure

and algorithm, no constraints on the design of individual robot control systems, and good simulation results make the proposed scheme attractive for many industrial applications.

CHAPTER VII

DIRECT CONTROL AND COORDINATION USING NEURAL NETWORKS

7.1 Introduction

In the previous chapters, we have developed a knowledge–based coordinator. The combination of the techniques of intelligent control and neural networks provides such advantages as a hierarchical structure, a simplified knowledge base and inference process, less *a priori* knowledge requirements on subsystems, and flexibility of adding more production rules. However, the potential capability of neural networks in control application does not seem to get a full play because neural networks are only used like an observer in the KBC. In this chapter, we will develop an NN–based controller to control a class of nonlinear systems and an NN–based coordinator for a tightly coupled multiple–system.

Many industrial control and coordination systems have difficulty achieving high performance with conventional control designs. For example, the main problems in process control are negative effects such as a long system–response delay, the dead zone and/or saturation of actuator mechanisms, and the nonlinear response of control valves. Process and measurement noises also degrade system performance. The dynamic property of a controlled plant may not be very complex, even though its

detailed structure and parameters may be unknown. However, when such a plant is put in operation, it will be difficult for the control system to achieve high performance due mainly to the negative effects mentioned above.

Contemporary industrial process control systems dominantly rely on PID–type controllers, though the hardware to implement control algorithms has been improved significantly in recent years. In addition to the difficulty in achieving high control quality, the fine tuning of the controller's parameters is a tedious task, requiring experts with knowledge both in control theory and process dynamics. Another example is the coordinated control of multiple robots cooperation. Each robot is a stand–alone device equipped with commercially–designed servo controllers. When more than one robot must cooperate to accomplish a common goal, in addition to the good behavior of each individual robot, their effective coordination is crucial to achieve the desired level of overall performance. This coordination problem is usually organized hierarchically. The low level includes the servo controllers which are designed independently of, and separately from, each other. The addition of a high–level coordinator should not require alteration of the internal structure and/or parameters of the low–level controllers. The main difficulties associated with this coordination problem come from nonlinear system dynamics, kinematic redundancy, MIMO, inaccurate system parameter values, and so on. To cope with the above problems/difficulties, new controllers (coordinators) should be developed. The goal of this chapter is to develop such a new controller (coordinator) using neural networks. Particularly, we shall focus on

1. industrial process control in the presence of the nonlinearity of dead zone and saturation, and the negative effects of long response delays and process noises,
2. the coordinated control of two robots holding an object, in which each robot

is equipped with commercially–designed servo controllers.

A controller is usually connected serially to the controlled plant under consideration. For a multilayer perceptron, the weights of the network need to be updated using the network's output error. For an NN–based controller, the NN's output is the control command of the system. However, when the NN is serially connected to a controlled plant, the network's output error is unknown, since the desired control action is unknown. This implies that the BP algorithm for training an NN cannot be applied to control problems directly. Therefore, one of the key problems in designing a direct NN–based controller is to develop an efficient training algorithm.

We note that most of the work mentioned in the survey (Section 2.3 of Chapter 2) is in the form of indirect adaptive control or has complex training methods and system structures, and none of them were developed to coordinate multiple systems. This fact was summarized in [NP90]:

> At present, methods for direct adjusting the control parameters based on the output error (between the plant and the reference model output) are not available. This is because the unknown nonlinear plant lies between the controller and the output error.

In contrast to the indirect adaptive control, in this chapter we will develop a direct adaptive controller and a coordinator. A simple algorithm is proposed based on the BP for a class of nonlinear systems typified by industrial process control applications and for a multiple–robot coordination problem. The proposed NN–based controller (coordinator) is trained by using the system's output errors directly with a little *a priori* knowledge of the controlled plant.

In Section 7.2, the control problem using NNs is stated formally, and the ba-

sic structure of the proposed NN–based controller (coordinator) is analyzed. The training algorithm is developed in Section 7.3, and the corresponding theorems are proved. Section 7.4 presents a procedure for designing the NN–based controller and addresses problem related to its implementation. Section 7.5 summarizes the simulation results of a temperature control system in a thermal power plant to test the proposed NN–based controller. This is a typical system with a long response delay, nonlinearity of dead zone and saturation, and process noise in process control. The performance of the NN–based controller is also compared with a PI controller. In Section 7.6, the coordinated control of two robots holding an object is presented, including the dynamics of the coordinated systems, specification of the desired forces, force error analysis, and the system structure with an NN–based coordinator. The proposed NN–based coordinator is evaluated for two 2–link robots holding an object via simulation, and the results are presented in Section 7.7. Section 7.8 is a summary of the chapter.

7.2 Problem Statement and the NN–based Controller

A controlled plant can be viewed as a mapping from the control input to the system output:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x},\ \boldsymbol{u},\ t), \qquad \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x},\ \boldsymbol{u},\ t),$$

where $\boldsymbol{x} \in \mathbf{R}^m$, $\boldsymbol{y} \in \mathbf{R}^n$, and $\boldsymbol{u} \in \mathbf{R}^{N_2}$ are system state, output and input, respectively. The controller of this plant, if exists, can be represented as a mapping from the system feedback and/or feedforward to control commands:

$$\boldsymbol{u} = \boldsymbol{c}(\boldsymbol{y},\ \boldsymbol{y}_d,\ t), \tag{7.1}$$

where $\boldsymbol{y}_d$ is the desired system output. As is usually the case, only the system output is assumed to be measured.

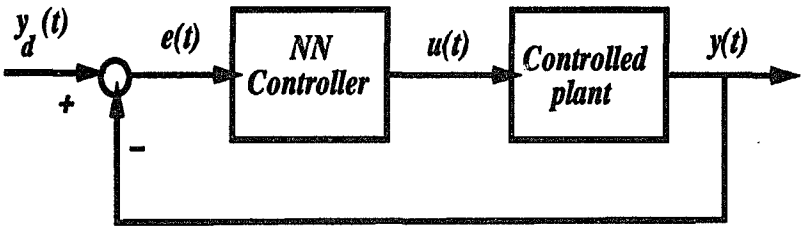


Figure 7.1: A control system with an NN–based controller.

We want to design an NN–based controller which will replace a conventional controller. In other words, the NN–based controller is cascaded with the controlled plant as shown in Fig. 7.1 and trained to learn the mapping in Eq. (7.1). The desired control input $\boldsymbol{u}_d(t)$, is required to produce the desired output $\boldsymbol{y}_d(t)$. The *system–output error* and the *control–input error* are then defined, respectively, by

$$\boldsymbol{e}_y(t) = \boldsymbol{y}_d(t) - \boldsymbol{y}(t), \qquad \text{and} \qquad \boldsymbol{e}_u(t) = \boldsymbol{u}_d(t) - \boldsymbol{u}(t).$$

The control–input error $\boldsymbol{e}_u(t)$, is also called the *network–output error*, since $\boldsymbol{u}(t)$ is the output of the NN–based controller. An NN is usually trained by minimizing the network–output error $\boldsymbol{e}_u(t)$. However, if the NN controller is cascaded in series with the controlled plant, then $\boldsymbol{e}_u(t)$ is not known, since the desired control input $\boldsymbol{u}_d(t)$ is unknown. So, the immediate problem in designing such an NN–based controller is how to train the NN.

As we have seen from the survey in Chapter 2, one of the most popular structures of neural networks is multilayer perceptron with BP algorithm. The BP algorithm is based on the gradient algorithm to minimize the network–output error and is derived from the special structure of the networks. In what follows, the BP algorithm for a three–layer perceptron is listed as a reference to see what is the problem using it as a controller/coordinator. Referring to Fig. 7.2, let θ_{1j} and θ_{2k} be the thresholds at the HIDDEN and the OUTPUT layer, respectively, where $1 \leq j \leq N_1$ and $1 \leq k \leq N_2$.

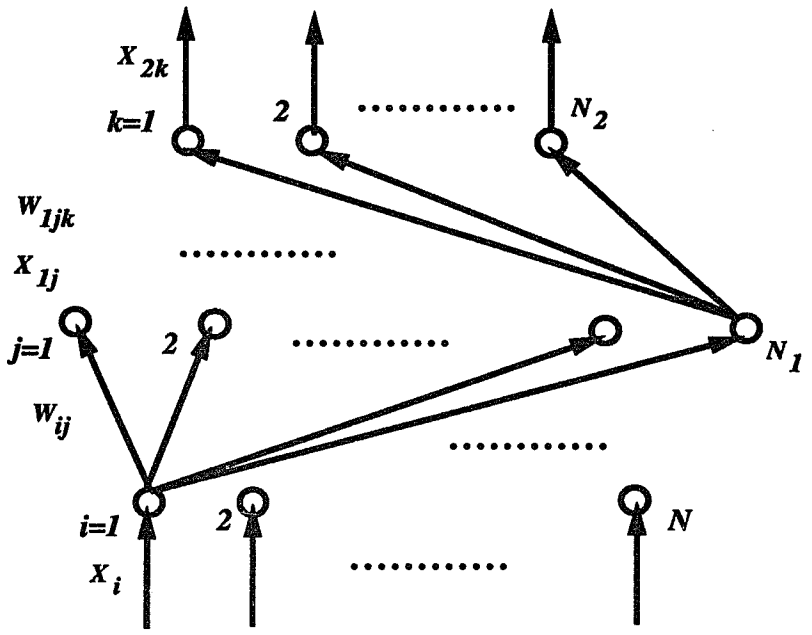


Figure 7.2: A multilayer perceptron used as an NN–based controller.

Then, the computation of the NN's output and updating of the NN's weights are summarized in the following five steps.

(1). Compute the output of the HIDDEN layer: X_{1j}

$$X_{1j}(t) = \frac{1}{1 + \exp\left(-O_{1j} - \theta_{1j}\right)}, \quad \text{where } O_{1j} = \sum_{i=1}^{N} W_{ij}\, X_i(t), \quad j = 1,\ 2,\ \cdots,\ N_1.$$

(2). Compute the output of the OUTPUT layer: X_{2k}

$$X_{2k}(t) = \frac{1}{1 + \exp\left(-O_{2k} - \theta_{2k}\right)}, \quad \text{where } O_{2k} = \sum_{j=1}^{N_1} W_{1jk}\, X_{1j}(t), \quad k = 1,\ 2,\ \cdots,\ N_2.$$

(3). Update the weights from the HIDDEN to the OUTPUT layer: W_{1jk}

$$W_{1jk}(t + \Delta t) = W_{1jk}(t) + \eta_1\, \delta_{1k}\, X_{1j}(t), \tag{7.2}$$

$$\text{where} \quad \delta_{1k} = \left(X^d_{2k}(t) - X_{2k}(t)\right)\, X_{2k}(t)\, \left(1 - X_{2k}(t)\right), \tag{7.3}$$

and X^d_{2k} is the desired value of X_{2k}.

(4). Update the weights from the INPUT to the HIDDEN layer: W_{ij}

$$W_{ij}(t + \Delta t) = W_{ij}(t) + \eta\, \delta_j X_i(t),$$

$$\text{where} \quad \delta_j \;=\; \left[\sum_{k=1}^{N_2} \delta_{1k}\, W_{1jk}(t+\Delta t)\right]\; X_{1j}(t)\;(1-X_{1j}(t))\,.$$

(5). Update the thresholds: θ_{2k} and θ_{1j}.

$$\theta_{2k}(t+\Delta t) = \theta_{2k}(t) + \eta_{1\theta}\;\delta_{1k}, \qquad \theta_{1j}(t+\Delta t) = \theta_{1j}(t) + \eta_{\theta}\;\delta_j, \qquad (7.4)$$

where η, η_1, η_θ, and $\eta_{1\theta} > 0$ are the gain factors.

In any control system design, it is desired to specify the system performance in terms of system–output errors, $\boldsymbol{e}_y(t) = \boldsymbol{y}_d(t) - \boldsymbol{y}(t)$, rather than the unknown network–output error $\boldsymbol{e}_u(t)$. To design such a controller using NNs, we adopt the basic principle of a multilayer perceptron with BP because of its ability of universal approximation and its convergent property based on the gradient algorithm [WM89]. The major obstacle to design such an NN–based controller is to train the NN using system–output errors, $\boldsymbol{e}_y(t)$, rather than the network–output errors $\boldsymbol{e}_u(t)$. This problem is solved in the next section.

7.3 Training an NN–based Controller with System–Output Errors

To derive the BP algorithm, the cost function of the network is defined as

$$E_u(t) = \frac{1}{2}\sum_{k=1}^{N_2} (e_{uk}(t))^2\,,$$

where $e_{uk}(t) = u_{kd}(t) - u_k(t)$ is the network–output error at the k-th node of the OUTPUT layer. As mentioned earlier, $E_u(t)$ is not available since $u_{kd}(t)$ is unknown for all k. Let the l-th component of the system–output error be defined by

$$e_{yl}(t) = y_{ld}(t) - y_l(t), \qquad l = 1,\ 2,\ \cdots,\ n.$$

Then, the cost function in terms of the system–output error is defined as

$$E_y(t) \;=\; \frac{1}{2}\sum_{l=1}^{n} (e_{yl}(t))^2 = \frac{1}{2}\sum_{l=1}^{n} (y_{ld}(t) - y_l(t))^2$$

$$= \frac{1}{2}\sum_{l=1}^{n}(\mathbf{G}_l(\boldsymbol{u}_d) - \mathbf{G}_l(\boldsymbol{u}))^2, \tag{7.5}$$

where $\mathbf{G}_l(\boldsymbol{u})$ is the l-th component of the dynamic system $\boldsymbol{y}(t) = \mathbf{G}(\boldsymbol{u}(t))$, $\boldsymbol{y}(t) = [y_1(t), \cdots, y_n(t)]^T$, and $\boldsymbol{u}(t) = [u_1(t), \cdots, u_{N_2}(t)]^T$. Eq. (7.5) is computable from the measurement of the system output. In other words, we know a function of the network–output error, though the detailed structure and parameters of the mapping $\mathbf{G}(\cdot)$ may not be known. We want to train the NN by minimizing the cost function Eq. (7.5).

Using the gradient algorithm, the weights from the HIDDEN to the OUTPUT layer are modified by

$$W_{1jk}(t + \Delta t) = W_{1jk}(t) + \Delta W_{1jk}, \tag{7.6}$$

$$\text{and setting} \qquad \Delta W_{1jk} \propto -\frac{\partial E_y(t)}{\partial W_{1jk}(t)}. \tag{7.7}$$

Noting that $u_k(t) = X_{2k}(t)$ in the NN–based controller[1], we get

$$\frac{\partial E_y(t)}{\partial W_{1jk}(t)} = -\sum_{l=1}^{n}\,(y_{ld}(t) - y_l(t))\,\frac{\partial y_l(t)}{\partial u_k(t)}\,\frac{\partial X_{2k}(t)}{\partial O_{2k}}\,\frac{\partial O_{2k}}{\partial W_{1jk}(t)}. \tag{7.8}$$

Since $\dfrac{\partial X_{2k}(t)}{\partial O_{2k}} = X_{2k}(t)\;(1 - X_{2k}(t))$ and $\dfrac{\partial O_{2k}}{\partial W_{1jk}(t)} = X_{1j}(t)$, Eq. (7.8) becomes

$$\frac{\partial E_y(t)}{\partial W_{1jk}(t)} = -\sum_{l=1}^{n}\,(y_{ld}(t) - y_l(t))\,\frac{\partial y_l(t)}{\partial u_k(t)}\,X_{2k}(t)\;(1 - X_{2k}(t))\;X_{1j}(t). \tag{7.9}$$

Substituting Eq. (7.9) into Eq. (7.7), one can get

$$\Delta W_{1jk}(t) = \eta_1^y\;\delta_{1k}^y\;X_{1j}(t), \tag{7.10}$$

$$\text{where} \quad \delta_{1k}^y = \sum_{l=1}^{n}\,(y_{ld}(t) - y_l(t))\,\frac{\partial y_l(t)}{\partial u_k(t)}\,X_{2k}(t)\,(1 - X_{2k}(t)), \tag{7.11}$$

[1]In fact, $X_{2k}(t)$ is the scaled value of $u_k(t)$. At this stage, it is assumed that the value of $u_k(t)$ is within the range of (0, 1). The scaling problem will be discussed later.

$\eta_1^y > 0$ is a gain factor. The only unknown in Eq. (7.11) is $\frac{\partial y_l(t)}{\partial u_k(t)}$, the $(l,\ k)$-th component of the Jacobian matrix of the controlled plant.

Recall that the network–output error at the k-th node of the OUTPUT layer is defined by

$$e_{uk}(t) = u_{kd}(t) - u_k(t). \tag{7.12}$$

Referring to Eq. (7.11), the component of system–output error contributed by the k-th control input is defined by

$$e_{sk}(t) = \sum_{l=1}^{n} \left(y_{ld}(t) - y_l(t)\right) \frac{\partial y_l(t)}{\partial u_k(t)}. \tag{7.13}$$

To apply the gradient algorithm, we have the following theorem.

Theorem 7.1: Suppose the system response delay corresponding to the k-th control input is d_0. To train the NN using the system–output error and ensure the convergence of the training algorithm, the necessary and sufficient condition is

$$sign\left(e_{sk}(t)\right) = sign\left(e_{uk}(t - d_0)\right). \tag{7.14}$$

Proof: In the gradient algorithm, the solution converges to a minimum of the cost function if and only if the search is made along the negative direction of the gradient of the cost function. BP is based on the gradient algorithm and listed in Eqs. (7.2) to (7.4). Because $u_{kd}(t) - u_k(t) = X_{2k}^d(t) - X_{2k}(t)$, Eq. (7.3) becomes

$$\delta_{1k} = e_{uk}(t)\ X_{2k}(t)\ \left(1 - X_{2k}(t)\right). \tag{7.15}$$

Substituting Eq. (7.13) into Eq. (7.11), we get

$$\delta_{1k}^y = e_{sk}(t)\ X_{2k}(t)\ \left(1 - X_{2k}(t)\right). \tag{7.16}$$

Because both Eqs. (7.15) and (7.16) are derived by applying the gradient algorithm, to ensure the convergence of the training algorithm given in Eqs. (7.6) and (7.10),

the necessary and sufficient condition is Eq. (7.14), when the system response delay is accounted for. □

The accurate value of $\left|\frac{\partial y_l(t)}{\partial u_k(t)}\right|$ is not important, because the step size can be adjusted by setting $\eta_s \equiv \eta_1^y \left|\frac{\partial y_l(t)}{\partial u_k(t)}\right|$. Certainly, this requires $\left|\frac{\partial y_l(t)}{\partial u_k(t)}\right| < \infty, \quad \forall\, t$. Therefore, if the sign of $\frac{\partial y_l(t)}{\partial u_k(t)}$ at each instant is known, then we get a simple algorithm to train the NN by using the system output error instead of the network output error. However, for general nonlinear systems, it is not easy to determine the sign of $\left|\frac{\partial y_l(t)}{\partial u_k(t)}\right|$ at each instant. Hence, in what follows, we shall develop a training algorithm for a class of systems with $\left|\frac{\partial y_l(t)}{\partial u_k(t)}\right| < \infty, \ \forall\, t$, and the following properties. Specially, in the next section, an NN–based controller is designed for a class of SISO systems, and the case of MIMO system is discussed in Sections 7.6 and 7.7.

7.4 Design of the NN–based Controller

For a SISO system, the training algorithm presented in the previous section can be simplified by using the definition of *system direction.*

Definition 1: If the system output monotonically increases (decreases) as the control input of a controlled plant increases, then the system is called *positive–responded* (*negative–responded*). Both positive–responded and negative–responded systems are called *monotone–responded.*

Definition 2: For a SISO system $y(t) = G(u(t))$, if the system is positive–responded (negative–responded), then the *system direction* is defined by $D(G) = 1$ $(D(G) = -1)$.

Definition 1 characterizes a class of systems. For example, a linear system is cascaded with an element of pure response delay, dead zone and/or saturation. Fortunately, there are many industrial process control systems that possess the property

of monotone–response. To train an NN–based controller for such a class of systems, we have the following theorem.

Theorem 7.2: For a SISO monotone–responded system, in order to train the NN–based controller in Fig. 7.2 using system–output error, the weights on the arcs from the HIDDEN to the OUTPUT layer are updated by

$$\begin{aligned} W_{1j1}(t+\Delta t) &= W_{1j1}(t) + \eta_1^y \, \delta_{11}^y \, X_{1j}(t), \qquad (7.17) \\ \text{where} \quad \delta_{11}^y &= (y_d(t) - y(t)) \; D(G) \; X_{21}(t) \; (1 - X_{21}(t)) . \end{aligned}$$

Proof: For a SISO system, Eqs. (7.12) and (7.13) are simplified to $e_u(t) = u_d(t) - u(t)$ and $e_s(t) = (y_d(t) - y(t)) \; \dfrac{\partial y(t)}{\partial u(t)}$. From Eq. (7.14), we get the condition of convergence: $sign\,(e_s(t)) = sign\,(e_u(t - d_0))$. If the system response delay is d, then for a positive–responded system

$$sign\,(u_d(t - d_0) - u(t - d_0)) = sign\,(y_d(t) - y(t)) . \qquad (7.18)$$

Similarly, for a negative–responded system we have

$$sign\,(u_d(t - d_0) - u(t - d_0)) = -sign\,(y_d(t) - y(t)) . \qquad (7.19)$$

From Eqs. (7.18) and (7.19), we conclude that the condition for convergence is

$$sign\,(u_d(t - d_0) - u(t - d_0)) = sign\,(y_d(t) - y(t)) \; D(G). \qquad (7.20)$$

Eq. (7.20) then implies that the corresponding training algorithm is based on Eq. (7.17). □

Figs. 7.1 and 7.2 show the basic structures of the system and the NN–based controller, respectively. For a SISO system, there is one node at the OUTPUT layer, that is, $N_2 = 1$. The choice of the NN's inputs should reflect the desired and actual

status of the controlled system. Therefore, the inputs of the NN–based controller are usually the system's desired and actual outputs, and tracking errors:

$$y_d(t) \ , \ y_d(t-\Delta t), \ \cdots, \ y_d(t-m_1\ \Delta t), \ \ y(t), \ y(t-\Delta t), \ \cdots, \ y(t-m_2\ \Delta t),$$

$$e_y(t) \ , \ e_y(t-\Delta t), \ \cdots, \ e_y(t-m_3\ \Delta t),$$

where m_1, m_2 and $m_3 > 0$ are integer constants, and $e_y(t) = y_d(t) - y(t)$. The number of the HIDDEN nodes depends on the controlled plant under consideration. However, selection of a suitable number may require extensive experiments.

Based on Theorem 7.2, the formulas for updating the weights from the INPUT to the HIDDEN layer and the thresholds are derived using the same procedure given in Section 7.3. The computation of the NN–based controller for a SISO system is then summarized as follows.

A. Compute the output of the HIDDEN layer: $X_{1j}(t)$.

$$X_{1j}(t) = \frac{1}{1+\exp\left(-O_{1j}-\theta_{1j}\right)}, \quad \text{where} \ \ O_{1j} = \sum_{i=1}^{N} W_{ij}\ X_i(t), \quad j = 1,\ 2,\ \cdots,\ N_1.$$

B. Compute the output of OUTPUT layer: $X_{21}(t)$.

$$X_{21}(t) = \frac{1}{1+\exp\left(-O_{21}-\theta_{21}\right)}, \quad \text{where} \ \ O_{21} = \sum_{j=1}^{N_1} W_{1j}\ X_{1j}(t).$$

C. Update the weights from HIDDEN to OUTPUT layer: $W_{1j1}(t)$.

$$\begin{aligned} W_{1j1}(t+\Delta t) &= W_{1j1}(t) + \eta_1^y\ \delta_{11}^y\ X_{1j}(t), \\ \text{where} \ \ \delta_{11}^y &= \left(y_d(t) - y(t)\right)\ D(G)\ X_{21}(t)\ \left(1 - X_{21}(t)\right). \end{aligned}$$

D. Update the weights from INPUT to HIDDEN layer: $W_{ij}(t)$.

$$\begin{aligned} W_{ij}(t+\Delta t) &= W_{ij}(t) + \eta^y\ \delta_j^y\ X_i(t), \\ \text{where} \ \ \delta_j^y &= \delta_{11}^y\ W_{1j1}\ X_{1j}(t)\ \left(1 - X_{1j}(t)\right) \end{aligned}$$

where η_1^y and $\eta^y > 0$ are the gain factors.

E. Update the thresholds: θ_{21} and θ_{1j}.

$$\theta_{21}(t + \Delta t) = \theta_{21}(t) + \eta_{1\theta}^y \ \delta_{11}^y, \qquad \theta_{1j}(t + \Delta t) = \theta_{1j}(t) + \eta_\theta^y \ \delta_j^y,$$

where $\eta_{1\theta}^y$ and $\eta_\theta^y > 0$ are the gain factors of the thresholds at the OUTPUT and the HIDDEN layer, respectively.

Another problem in designing such an NN–based controller is the choice of scaling factors. The sigmoid function in NN computation forces the NN outputs to be within the range of (0, 1), although the control input $u(t)$, is limited by the range of actuators, (U_{min}, U_{max}). Therefore, the NN outputs should coincide with, or be a little narrower than, the range of the actuator's limits. The output of the NN–based controller is then computed by

$$u(t) = X_{21}(t) \ (U_{max} - U_{min}) + U_{min}.$$

Generally, an NN works in the mode of *training — operation.* In other words, an NN is put in operation only after it is "well–trained." By "well–trained," we mean that the weights of the NN need not be modified any more. However, for a time–varying system, it is meaningless to say that an NN is "well–trained", since the system always changes with time. Thus, not updating the weights for a time–varying system may result in the system going out of control. It is therefore necessary to always update the weights of the NN–based controller. In other words, the weights of the NN–based controller should be updated but not in the mode of *training — operation*, though the updating may not be done during every sampling interval.

7.5 Simulation Results of a Temperature Control System

Many industrial process control systems can be characterized by a linear system cascaded with a nonlinear element as a result of dead zone and actuator limits, and/or a pure time delay caused by transportation delay and system response delay. To test the capability of the proposed NN–based controller, we conducted simulations while emphasizing the ability to overcome the negative effects of dead zone, saturation, long response delay, and process noise. The simulated system is a simplified temperature control system of a once–through boiler in a thermal power plant. The input is the variation of feedwater flow rate. The output is the variation of the temperature at the middle point where water becomes steam. The system is represented by an ARMAX model:

$$\mathbf{A}(z^{-1})\, y(k) = \mathbf{B}(z^{-1})\, u(k-d_0) + \mathbf{C}(z^{-1})\, \xi(k) \tag{7.21}$$

$$\begin{aligned} \text{where} \quad \mathbf{A}(z^{-1}) &= 1 - 0.45181\, z^{-1} - 0.47546\, z^{-2}, \\ \mathbf{B}(z^{-1}) &= -0.04560\, z^{-1} - 0.00404\, z^{-2}, \\ \mathbf{C}(z^{-1}) &= 1 - 0.35740\, z^{-1} - 0.03392\, z^{-2}, \\ d_0 &= 18 \text{ sampling intervals.} \end{aligned}$$

Here the sampling interval is chosen to be 8 seconds, $y(k)$ and $u(k)$ are the system output and control input at a discrete time k, respectively, and $\xi(k)$ is an uncorrelated random sequence with zero mean and variance R that represents the process noise. Note that this model is for simulation only. The NN–based controller has no knowledge about this system except its response direction.

A nonlinear element of dead zone and saturation is cascaded with the system Eq.

(7.21) to model an actuator, which is described by

$$u_a(t) = \begin{cases} U_{min} & \text{if } |u(t)| \geq dead\_zone \quad \text{and } U_{min} > u(t) \\ 0, & \text{if } |u(t)| < dead\_zone \\ u(t), & \text{if } |u(t)| \geq dead\_zone \quad \text{and } U_{min} \leq u(t) \leq U_{max} \\ U_{max}, & \text{if } |u(t)| \geq dead\_zone \quad \text{and } u(t) > U_{max} \ . \end{cases} \tag{7.22}$$

The dead zone and saturation are treated as unknown properties of the controlled plant. We want the NN–based controller to overcome their negative effects by NN's learning ability. Actually, since the system response direction will not be changed by adding dead zone and saturation, the NN–based controller should work well. Moreover, there is no special consideration for process noise in the design of NN–based controller, like other deterministic controller designs, though controllers have to be tested for the ability of noise rejection.

To reflect the status of the controlled system, the inputs of the NN–based controller are chosen as the desired system outputs and the output errors:

$$\begin{array}{lll} y_d(k), & y_d(k-1), & y_d(k-2), \\ y_d(k)-y(k), & y_d(k-1)-y(k-1), & y_d(k-2)-y(k-2). \end{array}$$

That is, there are six inputs at the INPUT layer of the NN-controller ($N = 6$). Note that the middle–point temperature system is in fact a high order system, though it can be approximately modeled by a low order linear system with a long pure time delay. Certainly, the NN–based controller is not used to model this high order controlled plant but to control it. So, it may be not necessary to use the same time delay of the controlled plant (18 sampling intervals in Eq. (7.21)) as its inputs. We also tested the NN–based controller with more delayed inputs. The results are not superior to those presented below. The number of the HIDDEN nodes is selected to be three ($N_1 = 3$). The overall system structure is sketched in Fig. 7.3.

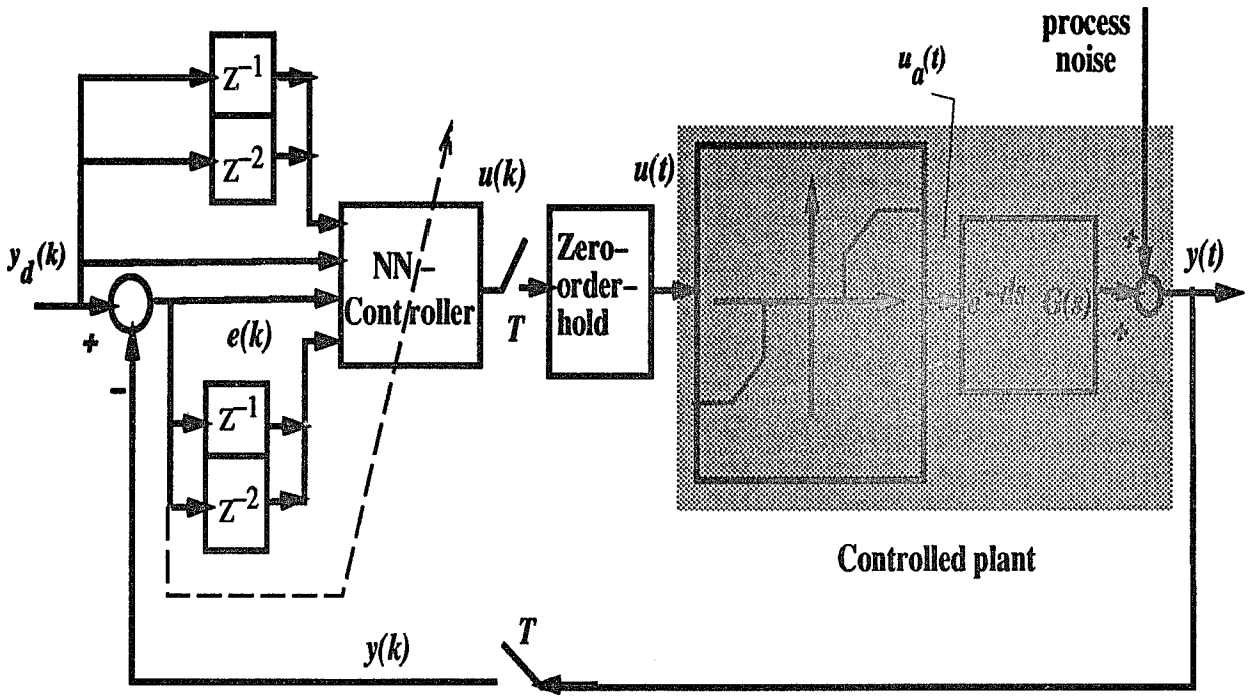


Figure 7.3: Structure of an NN–based control system.

The main simulation results are summarized below.

1. When $dead\_zone = 5.0$, $U_{max} = 10.0$, $U_{min} = -10.0$, and $R = 0.0$ (no process noise), the result is plotted in Fig. 7.4. The initial weights of the NN are selected randomly, and the NN weights converge within 150 sampling intervals.

2. When $dead\_zone = 7.0$, $U_{max} = 10.0$, $U_{min} = -10.0$, and $R = 0.0$, Figs. 7.5 and 7.6 present the system response and the corresponding control input, respectively. Obviously, a large dead zone affects the system performance severely, but the NN–based controller still works well.

3. When $dead\_zone = 5.0$, $U_{max} = 10.0$, $U_{min} = -10.0$, and $R = 0.5$ to test the ability of noise rejection, the desired and actual system output responses are plotted in Fig. 7.7. The corresponding control input and the process noise are shown in Fig. 7.8, where $n(k) = \xi(k) - 0.35740\ \xi(k-1) - 0.03392\ \xi(k-2)$.

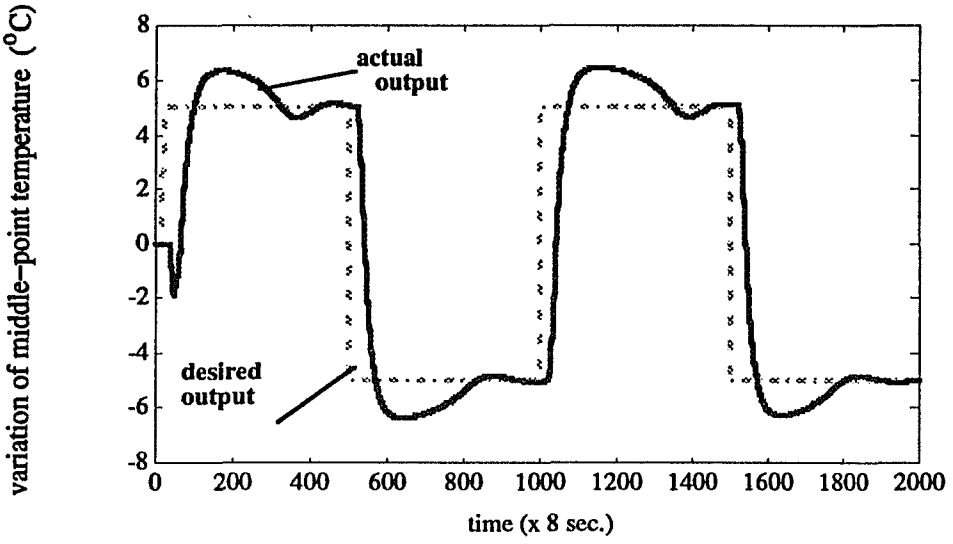


Figure 7.4: System output response with *dead_zone* = 5.0.

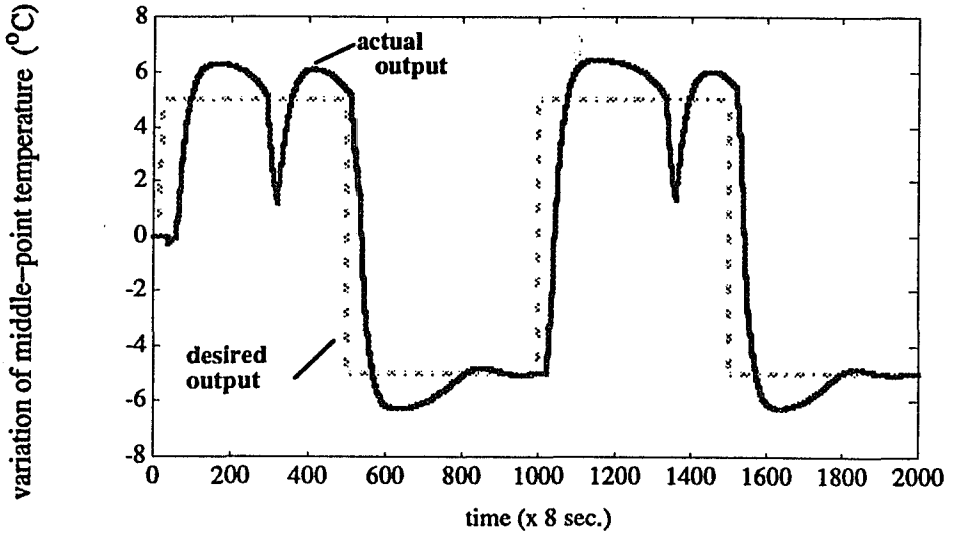


Figure 7.5: System output response with *dead_zone* = 7.0.

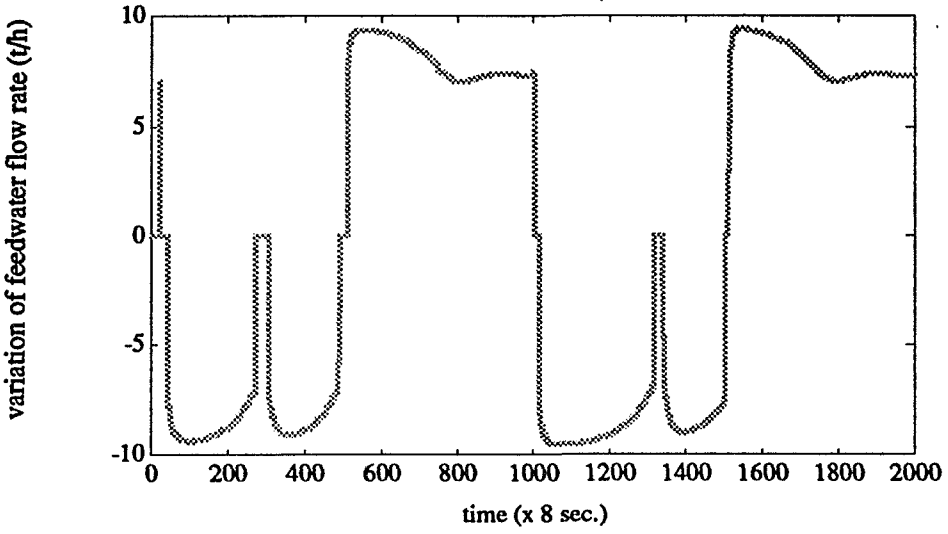


Figure 7.6: System control input with *dead_zone* = 7.0.

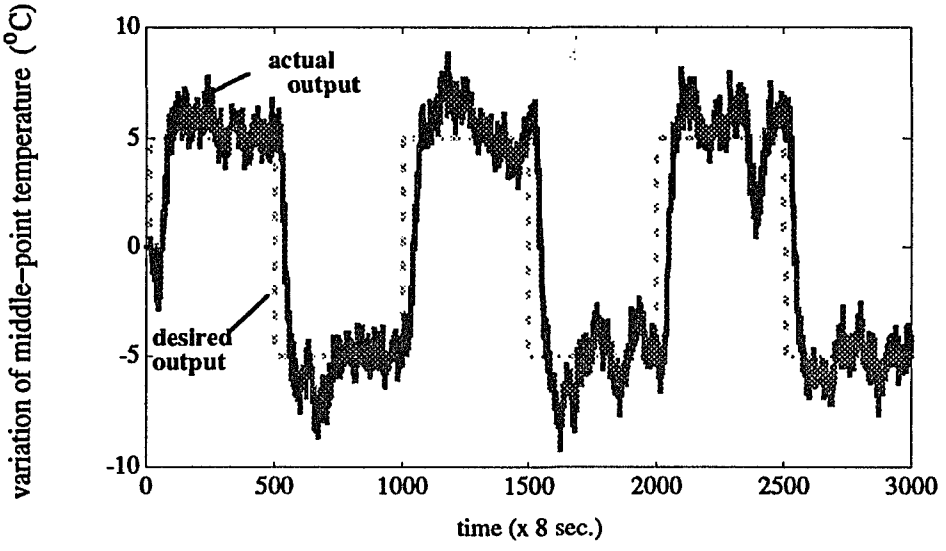


Figure 7.7: System output response with process noise $R = 0.5$.

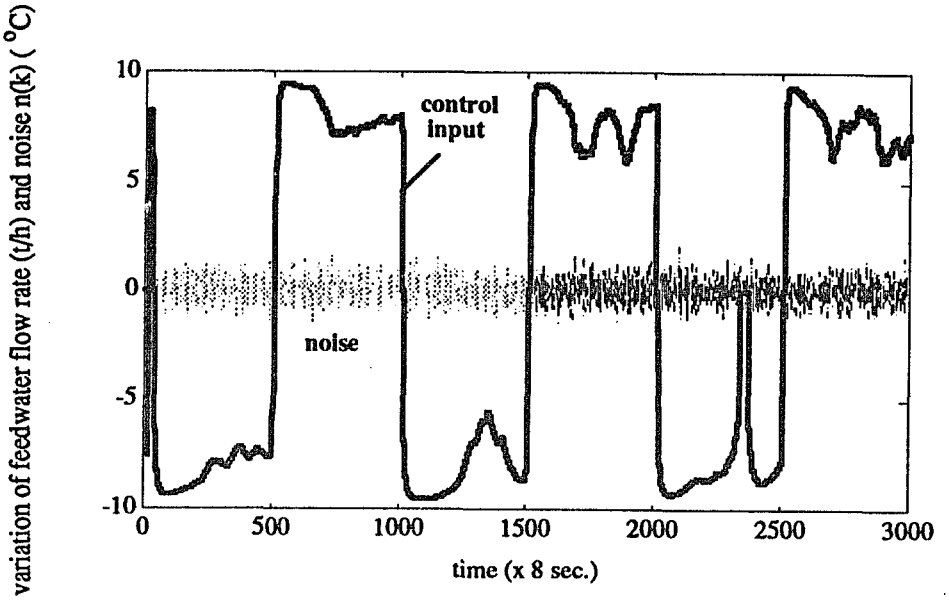


Figure 7.8: System control input and process noise with $R = 0.5$.

To compare the performance of the proposed NN–based controller with that of a PID–type controller, we have designed a PI controller for the middle–point temperature control system. The PI controller is

$$u(k) = -K_p\,(y_d(k) - y(k)) - K_I \sum_{i=0}^{k} (y_d(i) - y(i))$$

with $K_p = 2.2$ and $K_I = 0.3$. When $dead\_zone = 5.0$, $U_{max} = 10.0$, $U_{min} = -10.0$ and no process noise ($R = 0.0$), the system response controlled by the PI controller is plotted in Fig. 7.9. According to this figure, one should decrease K_p in order to reduce the oscillation. However, due to the effects of dead zone, one cannot make any notable improvement in the system performance. On the other hand, due to the effects of long time delay, increasing K_p will lead to an unstable response. When $dead\_zone = 5.0$, $U_{max} = 10.0$, $U_{min} = -10.0$ and $R = 0.5$, the result of the PI controller is plotted in Fig. 7.10. Comparing Fig. 7.9 with Fig. 7.4, and Fig. 7.10 with Fig. 7.7, we conclude that the performance of the NN–based controller is much

better than that of the PI controller. Actually, a PI controller cannot perform well for a system with a long time delay, dead zone, saturation, and process noise.

From the above simulation results, we conclude that the proposed NN–based controller performs well for this class of nonlinear systems. In the NN–based controller, the system–output error is computed from the measurements. As *a priori* knowledge, the system direction is easily obtained either from a step response experiment or from the physical property of a controlled plant. To test the need of Eq. (7.17), $-D(G)$ is used in the training algorithm, which instantly results in the NN's divergence.

The remaining problem is how to choose the number of the HIDDEN nodes. There is no systematic way to choose the number of the nodes at the HIDDEN layer(s) to approximate a given mapping. Therefore, selection of hidden nodes may depend on experiments. Fig. 7.11 shows the result using $N_1 = 6$, $dead\_zone = 5.0$, $U_{max} = 10.0$, $U_{min} = -10.0$ and $R = 0.0$. Comparing Fig. 7.11 with Fig. 7.4, one can see that adding more HIDDEN nodes does not improve the system performance. But adding more nodes will improve the system's reliability.

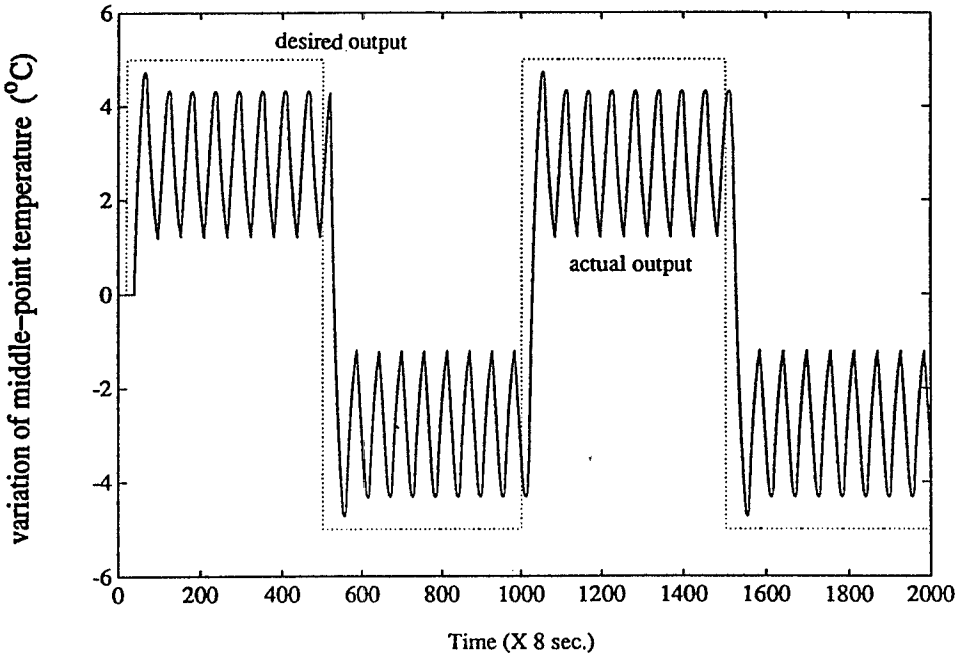


Figure 7.9: System response with a PI controller when $dead\_zone = 5.0$, and $R = 0.0$.

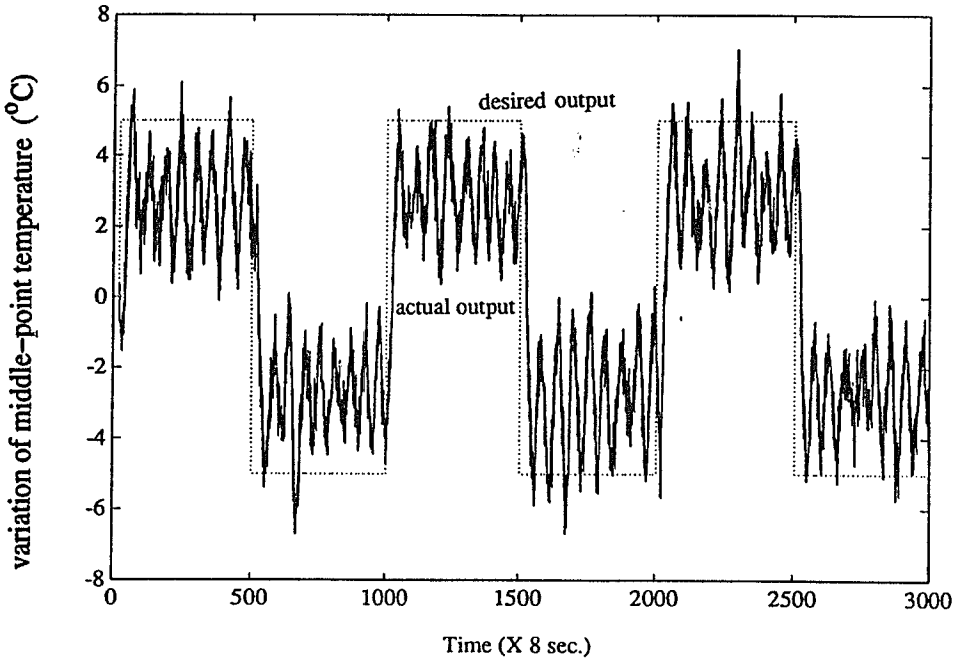


Figure 7.10: System response with a PI controller when $dead\_zone = 5.0$, and $R = 0.5$.

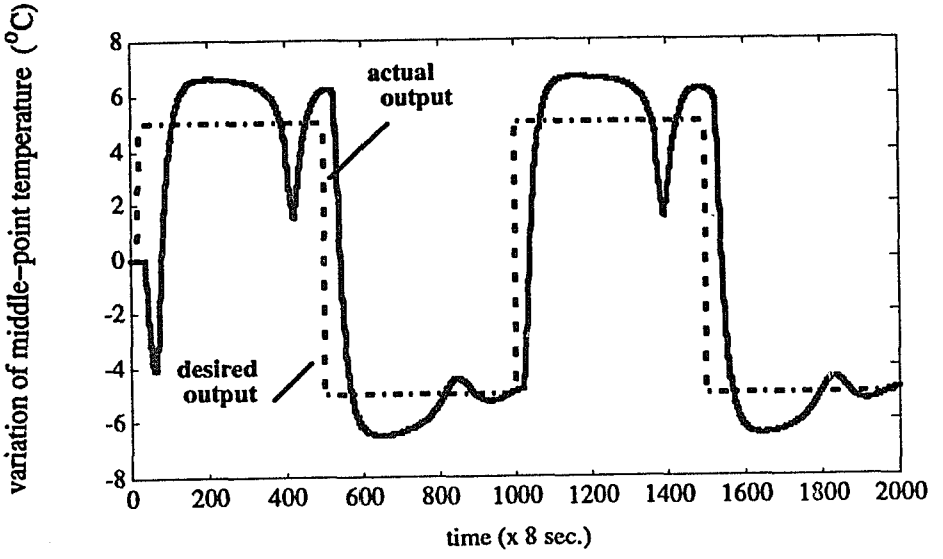


Figure 7.11: System output response with six hidden nodes.

7.6 Design of the NN–based Coordinator for Two 2–Link Robots

To test the proposed algorithm for multiple–system coordination, as an example, an NN–based coordinator (NNBC) is designed to coordinate two 2–link robots holding an object in this section. The basic configuration of this example is given in Fig. 5.5. The purpose is to investigate the suitability of the proposed algorithm. As we stated before, with a NNBC, the system forms a hierarchical structure, the high level is the NNBC, and the low–level subsystems include two robots each with a separately designed servo controllers.

Dynamics of the Coordinated Systems and Problem Statement

The detailed dynamics of the coordinated systems and the problem statement are presented in Section 5.5, and only the key points are summarized below.

Suppose the two robots hold the object firmly, then the motion of the object is

described by

$$m\ddot{\boldsymbol{P}} + m\boldsymbol{g} = \boldsymbol{f}, \qquad \boldsymbol{f} = \boldsymbol{f}_1 + \boldsymbol{f}_2, \tag{7.23}$$

where m is the mass of the object, $\boldsymbol{P}$ the position of the object in Cartesian space, $\boldsymbol{g}$ the gravitational acceleration, $\boldsymbol{f}_i = [f_{ix},\ f_{iy}]^T$ the external force exerted on the object by robot i.

Suppose two robots have an identical mechanical configuration, then the force–constrained dynamic equation of robot i in joint space is given by

$$\mathbf{H}(\boldsymbol{q}_i)\ddot{\boldsymbol{q}}_i + \mathbf{C}(\boldsymbol{q}_i,\ \dot{\boldsymbol{q}}_i)\ \dot{\boldsymbol{q}}_i + \mathbf{G}(\boldsymbol{q}_i) + \mathbf{J}_i^T\ \boldsymbol{f}_i = \boldsymbol{\tau}_i, \qquad i = 1,\ 2, \tag{7.24}$$

where $\boldsymbol{q}_i = [q_{i1},\ q_{i2}]^T$ and $\boldsymbol{\tau}_i = [\tau_{i1},\ \tau_{i2}]^T$ are the vectors of the joint position and torque of robot i, respectively. $\mathbf{J}_i$ is the Jacobian matrix, and the other terms are explained in Eq. 3.19 and the appendix.

In the proposed NNBC, the controlled joint torque consists of two parts

$$\boldsymbol{\tau}_i = \boldsymbol{\tau}_{ip} + \boldsymbol{\tau}_{ic}.$$

$\boldsymbol{\tau}_{ic}$ is contributed by the NNBC, and $\boldsymbol{\tau}_{ip}$ is given by a position controller

$$\boldsymbol{\tau}_{ip} = \hat{\mathbf{H}}\,(\ddot{\boldsymbol{q}}_{id} - \mathbf{K}_{Di}(\dot{\boldsymbol{q}}_i - \dot{\boldsymbol{q}}_{id}) - \mathbf{K}_{pi}(\boldsymbol{q}_i - \boldsymbol{q}_{id})) + \hat{\mathbf{h}}, \tag{7.25}$$

where $\hat{\mathbf{H}}$ and $\hat{\mathbf{h}}$ are the estimated values of $\mathbf{H}$ and $\mathbf{C}\ \dot{\boldsymbol{q}}_i + \mathbf{G}$, $\boldsymbol{q}_{id}$ is the desired value of $\boldsymbol{q}_i$, $\mathbf{K}_{Di}$ and $\mathbf{K}_{pi}$ are the controllers' gains.

As stated in Section 5.5, we want to coordinate the two robots moving the object while minimizing the internal force. In this section, we will design an NNBC to achieve this goal.

Specification of the Desired Forces and Force Error Analysis

Let the desired force sharing of the two robots is

$$\boldsymbol{f}_{1d} = \boldsymbol{\sigma}\ \boldsymbol{f}_d + \boldsymbol{f}_b, \qquad \boldsymbol{f}_{2d} = (\mathbf{I}_6 - \boldsymbol{\sigma})\boldsymbol{f}_d - \boldsymbol{f}_b, \tag{7.26}$$

where $\boldsymbol{\sigma}$ is a selection matrix, $\boldsymbol{\sigma} = diag[\sigma_1,\ \sigma_2]$, $0 \leq \sigma_j \leq 1,\ \ j = 1,\ 2$; $\boldsymbol{f}_b$ a bias force. Then the desired external and internal forces are

$$\boldsymbol{f}_d = \boldsymbol{f}_{1d} + \boldsymbol{f}_{2d} \quad \text{and} \quad \boldsymbol{f}_{Id} = \frac{1}{2}(\boldsymbol{f}_{1d} - \boldsymbol{f}_{2d}) = \boldsymbol{f}_b + \frac{1}{2}(2\boldsymbol{\sigma} - \mathbf{I}_6)\ \boldsymbol{f}_d.$$

Therefore, the desired external force $\boldsymbol{f}_d$, the selection matrix $\boldsymbol{\sigma}$ and the desired bias force $\boldsymbol{f}_b$ should be specified to compute the desired force exerted by each robot: $\boldsymbol{f}_{1d}$ and $\boldsymbol{f}_{2d}$.

Suppose the measured forces are $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$, then the actual external and internal forces exerted on the object are

$$\boldsymbol{f} = \boldsymbol{f}_1 + \boldsymbol{f}_2 \quad \text{and} \quad \boldsymbol{f}_I = \frac{1}{2}(\boldsymbol{f}_1 - \boldsymbol{f}_2).$$

Then, the force errors are

$$\boldsymbol{f}_{1e} = \boldsymbol{f}_{1d} - \boldsymbol{f}_1 = \boldsymbol{\sigma}\ \boldsymbol{f}_d + \boldsymbol{f}_b - \boldsymbol{f} + \boldsymbol{f}_2, \quad \text{and} \quad \boldsymbol{f}_{2e} = -(\boldsymbol{\sigma}\ \boldsymbol{f}_d + \boldsymbol{f}_b - \boldsymbol{f}_d + \boldsymbol{f}_2).$$

If the external force achieves its desired value, then we have $\boldsymbol{f}_{1e} + \boldsymbol{f}_{2e} = 0$. So, these force errors do not contribute to moving the object, that is, they are caused by internal force errors. This implies that if we design a controller based on $\boldsymbol{f}_{1e}$ to regulate the internal force and suppose the controller is a linear controller with control output $\boldsymbol{f}_1'$, then the control action acted on robot 2 should be $\boldsymbol{f}_2' = -\boldsymbol{f}_1'$ [Pit88]. Note that the force on end–effector and joint torque are related by

$$\boldsymbol{\tau}_i = \mathbf{J}_i^T\ \boldsymbol{f}_i.$$

Therefore, when the control action is transformed into the joint space, we usually have $\boldsymbol{\tau}_1 \neq -\boldsymbol{\tau}_2$ due to different Jacobian matrices.

Referring to Eqs. (7.24) and (7.25), if $\hat{\mathbf{H}} = \mathbf{H}$ and $\hat{\mathbf{h}} = \mathbf{h}$, then the closed–loop system can be written as

$$\boldsymbol{f}_i = (\mathbf{J}_i^T)^{-1}\ \mathbf{H}\,((\ddot{\boldsymbol{q}}_{id} - \ddot{\boldsymbol{q}}_i) + \mathbf{K}_{Di}(\dot{\boldsymbol{q}}_{id} - \dot{\boldsymbol{q}}_i) + \mathbf{K}_{pi}(\boldsymbol{q}_{id} - \boldsymbol{q}_i)) + (\mathbf{J}_i^T)^{-1}\ \boldsymbol{\tau}_{ic}.$$

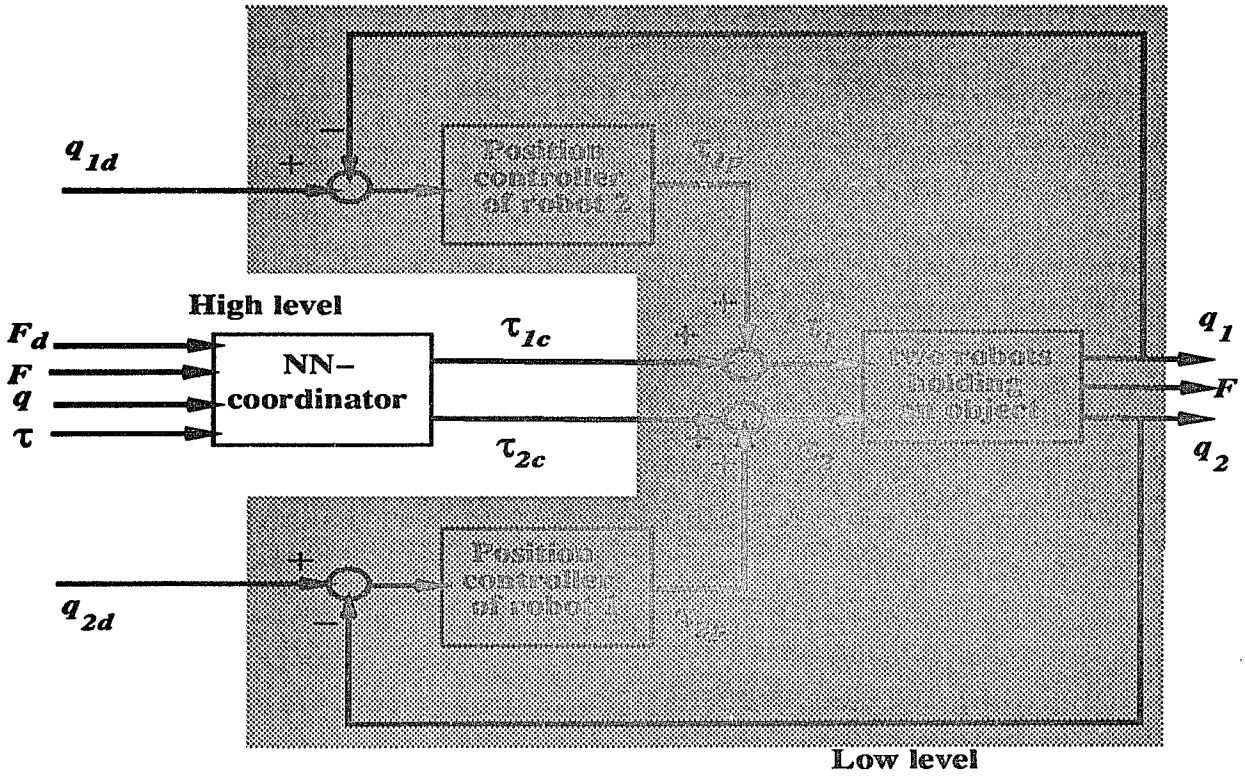


Figure 7.12: Coordinating two robots holding an object by an NNBC.

Since the desired external force is specified according to the desired trajectory, the desired external force can be achieved by a well–designed position controller. We can therefore design a coordinator so as to regulate the internal force by changing τ_{ic}.

Basic Structure of the System with an NN–based Coordinator

The basic structure of a two–robot system equipped with an NNBC is shown in Fig. 7.12. From the coordinator's point of view, the controlled plant is a mapping from the input torque $\tau_c = \left[\tau_{1c}^T,\ \tau_{2c}^T\right]^T$ to the forces exerted on the object F. This is an MIMO mapping $F = G(\tau_c)$ with time–varying property. We want to design an NNBC to directly control such a system using the theorems developed in Sections 3 and 4. For such an MIMO system, we define the direction matrix as follows.

Definition 3: For an MIMO system $F = G(\tau_c)$, the direction matrix of the

system is defined by

$$\mathbf{D(G)} \equiv sign\left(\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{\tau}_c}\right),$$

where the *sign* of a matrix $\mathbf{M}$ is defined as the matrix formed by the *sign* of the corresponding elements of $\mathbf{M}$.

When conditions such as the range of joint motion are posed on the system, it is possible to determine this matrix as shown in the example in the next section. From the force analysis, the following variables may be used as the inputs of the NNBC: the measured forces exerted on the object, $\boldsymbol{F}$, its desired value, $\boldsymbol{F}_d$, the measured joint positions of the two robots, $\boldsymbol{q} = \left[\boldsymbol{q}_1^T,\ \boldsymbol{q}_2^T\right]^T$, and the actual torque exerted on each joint of the two robots, $\boldsymbol{\tau}$. Obviously, all the inputs and outputs of this NNBC are vectors. For such a vector–structured multilayer perceptron, a vector form of the BP algorithm has been derived in Chapter 4. For the purpose of NNBC and counted in the direction matrix of the coordinated system, the main steps of the algorithm are summarized below.

All inputs and outputs of this NN are vectors, $\boldsymbol{X}_i \in \mathbf{R}^n$, $\boldsymbol{X}_{1j} \in \mathbf{R}^m$, and $\boldsymbol{X}_{2_1} \in \mathbf{R}^p$ are the output of INPUT, HIDDEN and OUTPUT layer, respectively, for $1 \leq i \leq N$, $1 \leq j \leq N_1$. The computation includes five steps

A. Compute the output of the HIDDEN layer $\boldsymbol{X}_{1j}$:

$$\begin{aligned}
\boldsymbol{X}_{1j} &\equiv [x_{1j1},\ \cdots,\ x_{1jm}]^T \\
&= \boldsymbol{f}_j(\boldsymbol{O}_{1j}) = \left[\frac{1}{1+\exp\left(-o_{1j1}-\theta_{1j1}\right)},\ \cdots,\ \frac{1}{1+\exp\left(-o_{1jm}-\theta_{1jm}\right)}\right]^T, \\
\boldsymbol{O}_{1j} &= \sum_{i=1}^{N} \mathbf{W}_{ij}\, \boldsymbol{X}_i, \quad j = 1,\ 2,\ \cdots,\ N_1,
\end{aligned}$$

where $\mathbf{W}_{ij} \in \mathbf{R}^{m\times n}$ is the weighting matrix from node i of the INPUT layer to node j of the HIDDEN layer, $\boldsymbol{f}_j : \mathbf{R}^m \rightarrow \mathbf{R}^m$ is defined as a sigmoid function of each component of a vector, and $\Theta_{1j} \equiv [\theta_{1j1},\ \cdots,\ \theta_{1jm}]^T$ is the threshold vector at node

j of the HIDDEN layer.

B. Compute the output of the OUTPUT layer $\boldsymbol{X}_{21}$

$$\begin{aligned}
\boldsymbol{X}_{21} &\equiv [x_{211}, \cdots, x_{21p}]^T \\
&= \boldsymbol{f}_1(\boldsymbol{O}_{21}) = \left[\frac{1}{1+\exp(-o_{211}-\theta_{211})}, \cdots, \frac{1}{1+\exp(-o_{21p}-\theta_{21p})}\right]^T, \\
\boldsymbol{O}_{21} &= \sum_{j=1}^{N_1} \mathbf{W}_{1j1}\, \boldsymbol{X}_{1j},
\end{aligned}$$

where $\mathbf{W}_{1j1} \in \mathbf{R}^{p\times m}$ is the weighting matrix from node j of the HIDDEN layer to the OUTPUT layer, $\boldsymbol{f}_1 : \mathbf{R}^p \to \mathbf{R}^p$, and $\Theta_{21} \equiv [\theta_{211}, \cdots, \theta_{21p}]^T$ is the threshold vector at the OUTPUT layer.

C. Update the weights from the HIDDEN to the OUTPUT layer $\mathbf{W}_{1j1}$:

$$\mathbf{W}_{1j1}(t+\Delta t) = \mathbf{W}_{1j1}(t) + \Delta\mathbf{W}_{1j1}, \qquad \text{where} \quad \Delta\mathbf{W}_{1j1} = \eta_1 \left[\boldsymbol{\delta}_{11}\, \mathrm{T}_1\right]^T,$$

$$\boldsymbol{\delta}_{11} = \left[\boldsymbol{X}_{21}^d - \boldsymbol{X}_{21}\right]^T \mathbf{D}(G)\; diag\left[x_{211}(1-x_{211}), \cdots, x_{21p}(1-x_{21p})\right],$$

and T_1 is a $p \times m \times p$ tensor, with the l-th matrix as

$$\mathbf{T}_{1l} = \begin{bmatrix} \mathbf{0} \\ \vdots \\ (\boldsymbol{X}_{1j})^T \\ \mathbf{0} \\ \vdots \end{bmatrix}^T \longleftarrow \text{at the } l\text{-th row}, \qquad l = 1,\ 2,\ \cdots,\ p.$$

D. Update the weights from the INPUT to the HIDDEN layer $\mathbf{W}_{ij}$:

$$\mathbf{W}_{ij}(t+\Delta t) = \mathbf{W}_{ij}(t) + \Delta\mathbf{W}_{ij}, \qquad \text{where} \quad \Delta\mathbf{W}_{ij} = \eta \left[\boldsymbol{\delta}_j\, \mathrm{T}\right]^T,$$

$$\boldsymbol{\delta}_j = \boldsymbol{\delta}_{11}\, \mathbf{W}_{1j1}(t+\Delta t)\; diag\left[x_{1j1}(1-x_{1j1}),\ x_{1j2}(1-x_{1j2}),\ \cdots,\ x_{1jm}(1-x_{1jm})\right],$$

and $\mathbf{T}$ is a $m \times n \times m$ tensor, with the l-th matrix as

$$\mathbf{T}_l = \begin{bmatrix} \mathbf{0} \\ \vdots \\ (\boldsymbol{X}_i)^T \\ \mathbf{0} \\ \vdots \end{bmatrix}^T \longleftarrow \text{at the } l\text{-th row}, \quad l = 1,\ 2,\ \cdots,\ m.$$

E. Update the thresholds at the OUTPUT and the HIDDEN layer Θ_{21}, and Θ_{1j}:

$$\Theta_{21}(t + \Delta t) = \Theta_{21}(t) + \Delta\Theta_{21}, \quad \text{where} \quad (\Delta\Theta_{21})^T = \eta_{1\theta}\ \boldsymbol{\delta}_{11}$$

$$\Theta_{1j}(t + \Delta t) = \Theta_{1j}(t) + \Delta\Theta_{1j}, \quad \text{where} \quad (\Delta\Theta_{1j})^T = \eta_{\theta}\ \boldsymbol{\delta}_{j}$$

η_1, η, $\eta_{1\theta}$ and $\eta_\theta > 0$ are the gain factors.

The problems of input–output scaling has been discussed in Section 7.4. In what follows, the design procedures are detailed with an example and tested via simulation.

7.7 Simulation Results of Two 2–Link Robots Holding an Object

referring to Fig. 5.5, the Cartesian frame is fixed at the base of robot 1, and the desired trajectories of the object and the robots' end–effectors are specified relative to this frame. The task is to move the object forward and backward in X direction while keeping the height in Y direction constant. The desired trajectory which is selected by a high–level planner is to move the object in X direction from an initial position to a final position (for one meter distance) in five seconds, and then move back to the initial position. The desired velocity and acceleration of the object are zero at both the initial and the final positions. The kinematic and dynamic parameters of

the robots are presented in Table 3.1. The sampling interval is 10 ms. The selection matrix is set to $\boldsymbol{\sigma} = diag\,[0.5,\ 0.5]$ and the bias force $\boldsymbol{f}_b = 0$. Each robot is position controlled with the controller in Eq. (7.25). Note that the NNBC has no knowledge about the dynamics of the coordinated robots except the direction matrix which is derived as follows.

Let $\boldsymbol{F} = [f_{1x},\ f_{1y},\ f_{2x},\ f_{2y}]^T$ be the forces exerted on the object by each robot in X and Y directions, and $\boldsymbol{\tau}_c = [\tau_{11c},\ \tau_{12c},\ \tau_{21c},\ \tau_{22c}]^T$ be the torque exerted on each joint of the two robots by the NNBC. Then the direction matrix is defined by

$$\mathbf{D} = sign\left(\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{\tau}_c}\right) = sign\left(\begin{bmatrix} \frac{\partial f_{1x}}{\partial \tau_{11c}} & \frac{\partial f_{1x}}{\partial \tau_{12c}} & \frac{\partial f_{1x}}{\partial \tau_{21c}} & \frac{\partial f_{1x}}{\partial \tau_{22c}} \\ \frac{\partial f_{1y}}{\partial \tau_{11c}} & \frac{\partial f_{1y}}{\partial \tau_{12c}} & \frac{\partial f_{1y}}{\partial \tau_{21c}} & \frac{\partial f_{1y}}{\partial \tau_{22c}} \\ \frac{\partial f_{2x}}{\partial \tau_{11c}} & \frac{\partial f_{2x}}{\partial \tau_{12c}} & \frac{\partial f_{2x}}{\partial \tau_{21c}} & \frac{\partial f_{2x}}{\partial \tau_{22c}} \\ \frac{\partial f_{2y}}{\partial \tau_{11c}} & \frac{\partial f_{2y}}{\partial \tau_{12c}} & \frac{\partial f_{2y}}{\partial \tau_{21c}} & \frac{\partial f_{2y}}{\partial \tau_{22c}} \end{bmatrix}\right).$$

From the configuration shown in Fig. 5.5, we can assume that the limitation of the joint angles are

$$0° < q_{11} < 180°, \quad -180° < q_{12} < 0°,$$

$$0° < q_{21} < 180°, \quad 0° < q_{22} < 180°.$$

Then the direction matrix can be determined as

$$\mathbf{D} = sign\left(\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{\tau}_c}\right) = \begin{bmatrix} -1 & -1 & +1 & +1 \\ +1 & +1 & +1 & +1 \\ -1 & -1 & +1 & +1 \\ +1 & +1 & +1 & +1 \end{bmatrix}.$$

This will be used in the computation of the NNBC. For this example, a three–layer perceptron is used. There are four INPUT nodes with inputs

$$\boldsymbol{q} = [q_{11},\ q_{12},\ q_{21},\ q_{22}]^T, \qquad \boldsymbol{F} = [f_{1x},\ f_{1y},\ f_{2x},\ f_{2y}]^T,$$
$$\boldsymbol{F}_d = [f_{1xd},\ f_{1yd},\ f_{2xd},\ f_{2yd}]^T \qquad \boldsymbol{\tau} = [\tau_{11},\ \tau_{12},\ \tau_{21},\ \tau_{22}]^T,$$

which reflect the desired and actual status of the coordinated system. The OUTPUT layer has one node with output

$$\boldsymbol{\tau}_c = \left[\boldsymbol{\tau}_{1c}^T,\ \boldsymbol{\tau}_{2c}^T\right]^T = [\tau_{11c},\ \tau_{12c},\ \tau_{21c},\ \tau_{22c}]^T.$$

To evaluated the performance of the proposed scheme, the NNBC is tested via simulation and the results are summarized below.

Suppose the mass of the object is 5 kg, without the NNBC, the internal force error in X direction is plotted in Fig. 7.13. By adding the NNBC with 15 hidden nodes, the the performance is greatly improved as shown in Fig. 7.14. The RMS error of the internal force in X direction is reduced by 94.6%. In Y direction, the RMS internal force error is reduced by 46.2%, though the internal force error is small enough due to no motion in this direction. Moreover, both the external force error and the position tracking error are kept almost the same as those without the coordinator. The detailed results are summarized in Table 7.1.

If the mass of the object is increased to $10kg$, the NNBC also works well with 20 hidden nodes. The internal force error in X direction is reduced by 89.8%, as shown in Figs. 7.15 and 7.16, and Table 7.2.

The remaining problems include:

- Choice of the number of HIDDEN–layer nodes. There is no systematic way to choose the number of the nodes at the HIDDEN layer(s) to approximate a given mapping. As shown in Section 7.5, adding more HIDDEN–layer nodes may not always improve the system performance.

Sample intervals		RMS errors of internal force (N)	
		without the NNBC	with the NNBC
0 — 1000	at X direction	9.58411	2.01236
	at Y direction	0.93142	0.51720
1001 — 2000	at X direction	9.57103	0.51404
	at Y direction	0.92337	0.49638
2001 — 3000	at X direction	9.57048	0.51248
	at Y direction	0.92337	0.49629
Sample intervals		RMS errors of external force (N)	
		without the NNBC	with the NNBC
0 — 1000	at X direction	0.72164	0.81624
	at Y direction	2.54845	3.53886
1001 — 2000	at X direction	0.34370	0.36121
	at Y direction	0.01436	0.01104
2001 — 3000	at X direction	0.34370	0.36120
	at Y direction	0.01436	0.01105
Sample intervals		RMS tracking errors of object's position (m)	
		without the NNBC	with the NNBC
0 — 1000	at X direction	0.03694	0.03772
	at Y direction	0.05733	0.00692
1001 — 2000	at X direction	0.03694	0.03773
	at Y direction	0.05759	0.00312
2001 — 3000	at X direction	0.03694	0.03773
	at Y direction	0.05759	0.00312

Table 7.1: RMS errors when $mass = 5kg$.

Sample intervals		RMS errors of internal force (N)	
		without the NNBC	with the NNBC
0 — 1000	at X direction	15.96137	3.75179
	at Y direction	1.12425	1.11996
1001 — 2000	at X direction	16.08066	1.89737
	at Y direction	1.10789	1.11099
2001 — 3000	at X direction	16.07986	1.84620
	at Y direction	1.10790	1.10275
Sample intervals		RMS errors of external force (N)	
		without the NNBC	with the NNBC
0 — 1000	at X direction	1.40957	1.84460
	at Y direction	6.67420	9.57597
1001 — 2000	at X direction	0.67540	0.88516
	at Y direction	0.06293	0.26797
2001 — 3000	at X direction	0.67540	0.85093
	at Y direction	0.06294	0.13169
Sample intervals		RMS tracking errors of object's position (m)	
		without the NNBC	with the NNBC
0 — 1000	at X direction	0.03696	0.03903
	at Y direction	0.11624	0.01390
1001 — 2000	at X direction	0.03696	0.03896
	at Y direction	0.11669	0.00657
2001 — 3000	at X direction	0.03696	0.03882
	at Y direction	0.11669	0.00652

Table 7.2: RMS errors when $mass = 10kg$.

- Start the NN–based controller (coordinator). Since the initial values of the weights are random numbers, the learning period may result in a large oscillation of the system output. This may be unacceptable for a certain controlled plant even for an open–loop stable system.

7.8 Summary

To handle difficult control and coordination problems, we developed a direct controller and a coordinator with neural networks. Particularly, the NN–based controller aims to handle industrial process control systems in which the negative effects of a long system response delay, nonlinear elements with dead zone and/or saturation, and process noises are the main obstacles in achieving high performance. The proposed NN–based controller can replace conventional controllers, and has overcome all of the problems mentioned above. The NNBC is applied to the coordinated control of two robots holding an object. Such a coordinated system is organized hierarchically, where the high level is the NNBC and the low level is the coordinated robots. It is assumed that each robot is a stand–alone device equipped with a commercially designed (perhaps by different vendors) servo controller. The internal structure and/or parameters of the low–level subsystems are not affected by adding the NNBC. This implies that some industrial robots could be coordinated to perform more sophisticated tasks than originally intended.

In contrast to the scheme of indirect adaptive control [NP90], the proposed scheme enables the NN to be trained with system–output errors, rather than the network–output errors. The training algorithm is derived based on BP. However, in the BP algorithm, it is required to modify the weights by network–output error which is not known when a multilayer perceptron is cascaded in series to the controlled

plant. Therefore, the proposed algorithm enhances the NN's ability to handle a wider range of control applications. A detailed analysis of the algorithm was presented and the associated theorems were proved. The only *a priori* knowledge about the controlled plant is the direction of its response, which is usually easy to determine for a SISO system. The direction matrix of an MIMO system can be determined, if some system constraints are imposed. Extensive simulations have been carried out and the results are shown to be quite promising. Good performance, a simple structure and algorithm, and the potential for fault tolerance make the proposed NN–based controller and the NNBC attractive for industrial application.

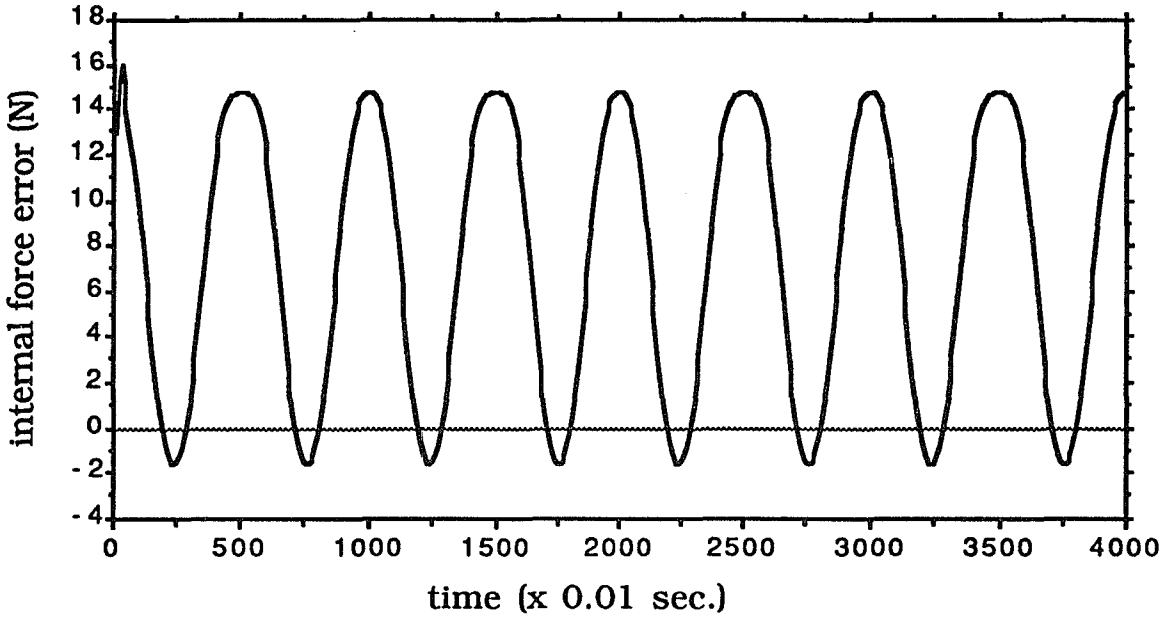


Figure 7.13: The internal force error in X direction, without the NNBC.

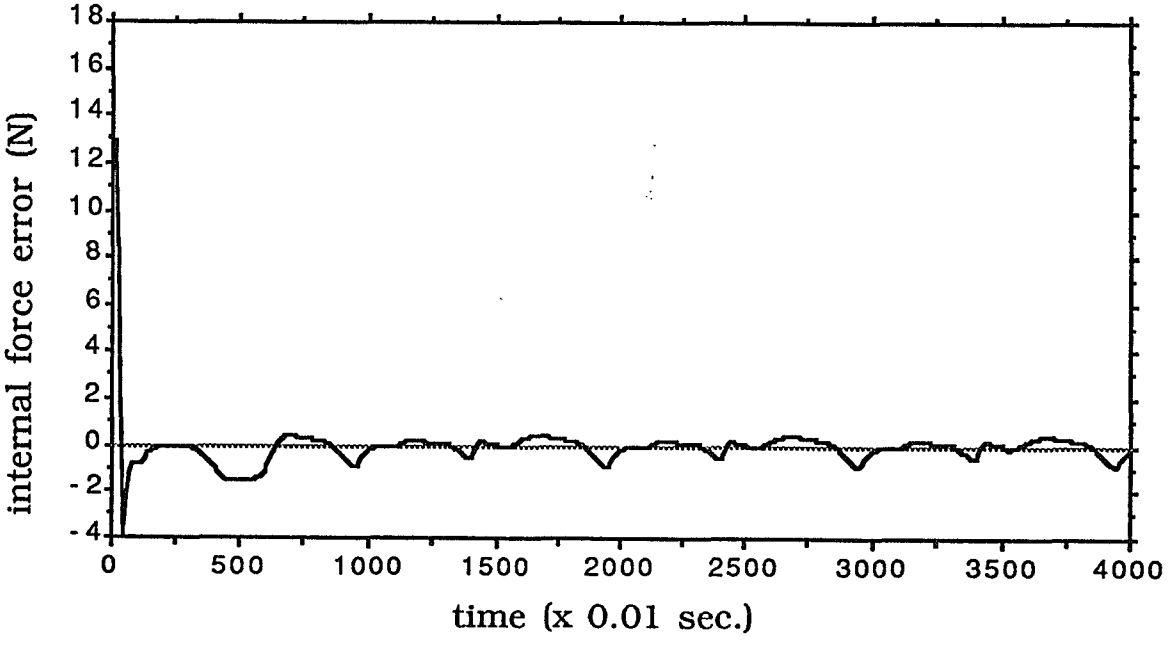


Figure 7.14: The internal force error in X direction, with the NNBC.

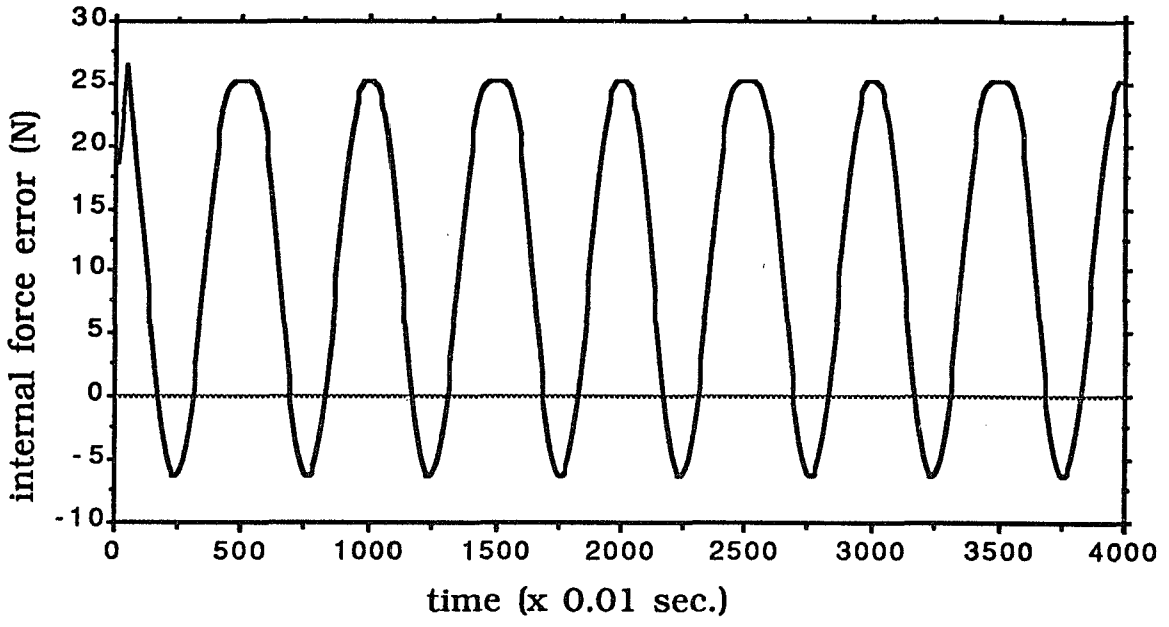


Figure 7.15: The internal force error as $m = 10\ kg$, without the NNBC.

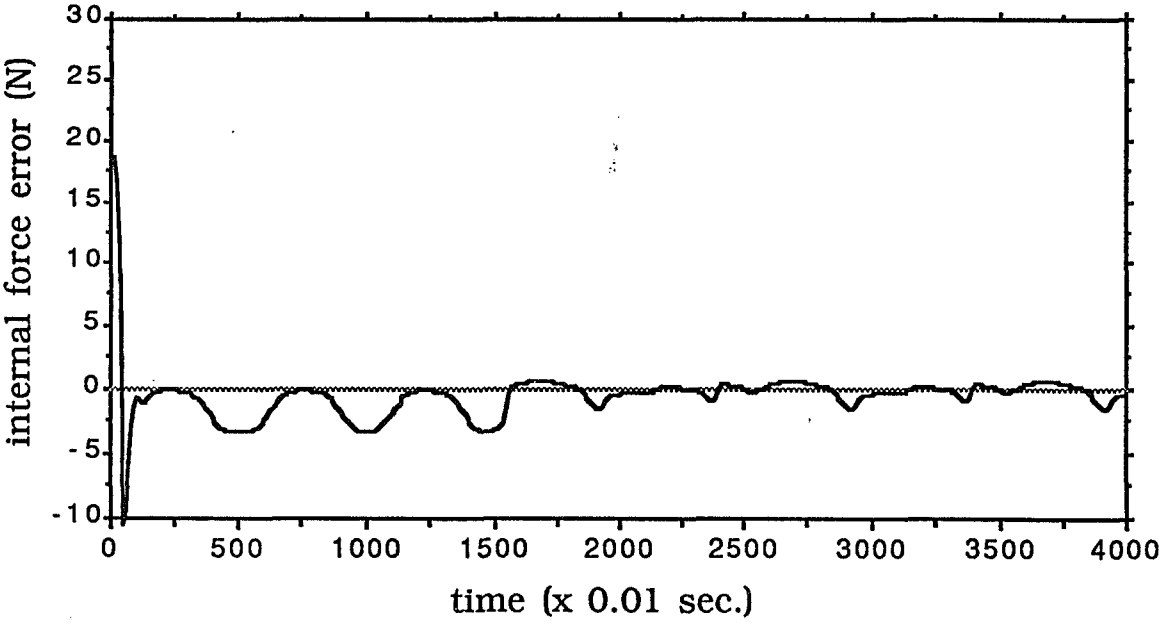


Figure 7.16: The internal force error as $m = 10\ kg$, with the NNBC.

CHAPTER VIII

PERFORMANCE EVALUATION OF REAL–TIME CONTROL AND COORDINATION SYSTEMS

8.1 Introduction

A real–time, digital, control computer or *controller computer* can be thought of as a three–stage pipe: data acquisition from sensors, data processing to generate control/display commands, and outputting the results to actuators/display devices. Although each of the three stages will take time to complete, this chapter is only concerned with the time taken by the most complicated stage, data processing, since the other two are much simpler and more static. More precisely, the amount of time taken to execute programs that implement control algorithms — called the *computing time delay* — is the subject of this chapter.

A controller computer implements the control algorithms by executing a sequence of instructions. Unlike analog control systems, the reliability of a digital control system depends not only on the MTBF (mean time between failures) of the controller hardware and software, but also on the delay in executing control algorithms on the controller computer. The execution time for a control algorithm is defined as the period from its trigger to generation of a corresponding control command. It is an extra

time delay that is introduced to the feedback loop in a controlled system. Because of the existence of conditional branches, resource sharing delays, and processing exceptions, the execution time for a given control algorithm, or the computing time delay, is usually a continuous, random variable which is usually smaller than the sampling interval. Controllers are usually designed without considering their actual implementation. Thus, it is very important to analyze the effects of computing time delay on control system performance when control algorithms are implemented on digital computers.

The computing time delay is quite *different* from the usual system time delay, and cannot be taken care of prior to putting a system in use due to its randomness caused, for example, by the data–dependent branches and loop counts in a program that implements the control algorithm under consideration.

When the computing time delay is long relative to the sampling interval (but small relative to the mission lifetime), it may seriously affect control system performance. Depending on the magnitude of computing time delay relative to the sampling interval, its effects on the control system are classified into either a *delay* or *loss* problem. To be more precise, let ξ and T_s denote the computing time delay and the sampling interval, respectively. A delay problem results when $0 < \xi < T_s$, and the loss problem occurs when $\xi \geq T_s$. The former represents the undesirable effects (for example, in terms of operational cost or energy) caused by a computing time delay which is nonzero but smaller than the deadline (that is, the beginning of the next sampling interval) of a control algorithm or task,[1] while the latter represents the case of no update of the control output for one or more sampling intervals.

To implement a control algorithm on a digital computer, the sampling rate must

[1]The terms "control task" and "control algorithm" will henceforth be used interchangeably.

be chosen carefully by not only satisfying the conditions of the Shannon's sampling theorem, but also by achieving the desired performance. A good example of this can be found in [Kho87] where a series of robot control experiments were conducted for different sampling rates while keeping the controller gains fixed. A higher sampling rate is shown to imply improved performance and higher stiffness (or disturbance rejection property). However, increasing the sampling rate will make the computing time delay effects more pronounced on control system performance. These effects cannot be neglected, especially when the time constant of the plant is short and the order of the plant is high [Mit85]. To remedy this problem, an optimal state feedback control law was proposed by [Mit85]. Instead of using the current–state feedback $u(k) = -\boldsymbol{K}\boldsymbol{x}(k)$, the control input is formed by $u(k) = -\boldsymbol{K}\mathbf{A}\boldsymbol{x}(k-1) - \boldsymbol{K}\mathbf{B}u(k-1)$, that is, the computing time delay is approximated to be one sampling interval. The sampling period, T_s, is chosen to be the same as the computation time of the control algorithm. In [BDG86], the computing time delay is represented as a delayed state measurement, and an averaging A/D device is used for the measurement. Then for an LQG–type, sampled–data regulator problem, an equivalent discrete–time problem is shown to have an increased system order. A design procedure was proposed there for this equivalent discrete–time problem.

However, the work in both [Mit85] and [BDG86] did not consider the randomness of the computing time delay. Approximating the computing time delay with one or more sampling intervals and incorporating it into controller design was the basic idea used there. As mentioned earlier, the computing time delay problem must account for the random effects of data–dependent conditional branches/loops and the unpredictable delays in sharing resources during the execution of control algorithms. So, estimating the maximum delay or assuming the computing time delay to be

constant is neither realistic nor possible.

The magnitude of the computing time delay and the number of output losses that the controlled system can tolerate are important indices to evaluate the performance of any digital control algorithm. These indices will inherently change with the type of control algorithms and systems under consideration. To demonstrate the importance of the computing time delay, we shall evaluate this index for typical real–time control systems — robot control and coordination systems, which are briefly described below.

Numerous robot control algorithms have been proposed and their computing time delays estimated. However, to our best knowledge, none of these has analyzed explicitly the effects of computing time delay on control system performance. For example, an adaptive control algorithm based on the computed torque algorithm is reported to require 17 ms on an MC68000 CPU [LL84], and the self–tuning predicted control needs 6.7 ms on an NS32132 CPU [CS88]. These results not only indicate the need of high–speed CPUs, but also raise an interesting question: can the system tolerate this extra computing time delay ? A robot control system is usually evaluated on the basis of tracking accuracy, repetition error, and motion speed. For a nonlinear, time–varying system like a robot, the effects of computing time delay on its performance become significant enough to warrant a careful investigation of various control algorithms before using them. This is also true for all other time–critical control systems such as aircraft and life–support systems. (See [SKL85] for an example of aircraft landing.)

In recent years, the application of artificial intelligence and neural networks has received considerable attention in control and robotics communities as surveyed in Chapter 2. In a knowledge–based control system, control actions are usually determined either by searching its knowledge base or looking up tables. This, especially

in the case of a heuristic search process, may require a significant amount of time relative to the sampling interval. Unlike a controller based on numerical computations, the time for symbolic reasoning and heuristic searches may vary with the operational conditions of the system. In an NN–based control system, if the neural network is implemented in software, then the computing time delay problem becomes very significant, since NN computation may take much longer time than conventional control algorithms. Moreover, loss of control outputs affects not only the system output at the instant of loss, but also the updating the NN's weights, which may, in turn, affect system performance. Therefore, both the delay problem and loss problem should be analyzed for knowledge–based and NN–based systems.

We shall focus on analyzing the effects of computing time delay on the performance of control and coordination systems. Using examples, we will also show how a specific control system is evaluated in terms of computing time delay. In Section 8.2, we first review the basic concepts and definitions related to real–time, digital control systems which were introduced in [SKL85]. Then, we address the generic problem of analyzing the effects of computing time delay on control system performance. A generic criterion for the qualitative analysis of computing time delay effects is derived, and a common misconception in handling computing time delay is corrected. We present in Section 8.3 both qualitative and quantitative analyses of the computing time delay effects on a robot control system. Upper bounds of computing time delay are derived with respect to system stability and system performance. These upper bounds can be used as an extra constraint on controller design or an index for selection of a CPU to implement a control algorithm. For a typical coordination problem — two robots holding an object — the effects of computing time delay associated with the knowledge–based coordinator and the NN–based coordinator developed in

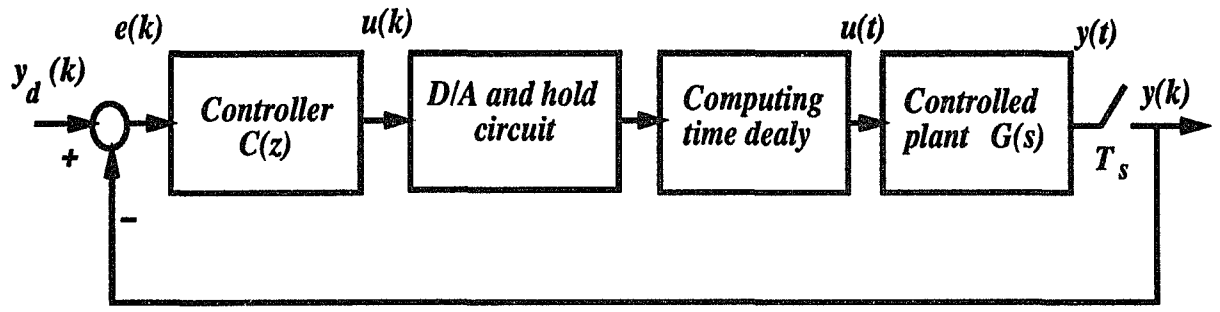


Figure 8.1: A digital control system in presence of computing time delay.

the previous chapters are investigated in Sections 8.4 and 8.5, respectively. Section 8.6 is a summary of the chapter.

8.2 Effects of Computing Time Delay on a Control System

The basic concepts and performance measures proposed in [SKL85] are best suited for the analysis of the effects of computing time delay on control system performance, because they are based on task completion times. For completeness, some of the basic concepts in [SKL85] are briefly described. Then, a generic analysis of the effects of computing time delay is presented along with necessary conditions for system stability.

8.2.1 Performance Measures in the Presence of Computing Time Delay

As mentioned earlier, we are interested in analyzing the effects of computing time delay that results from the implementation of a "well-designed" control algorithm on a digital computer. (By "well-designed", we mean that the system is stable and the effects of discretization are accounted for.) The presence of the computing time delay in a control system can be represented by a delay element after the D/A converter and hold circuit, as shown in Fig. 8.1. Hence, the analysis of the effects of computing time delay must be done in a continuous-time domain. Note that, due to

its randomness, the computing time delay is totally different from other usual factors, such as the effects of discretization and system delay, which are *not* the subject of this chapter.

Let $\mathbf{X} \subset \mathbf{R}^n$ denote the state space and $\boldsymbol{x}(t) \in \mathbf{X}$ the state of the controlled system at time t. Evolution of states from time t_0 in the presence of a nonzero computing time delay, ξ, is represented by

$$\boldsymbol{x}(t) = \Phi(t,\ t_0,\ \boldsymbol{x}(t_0),\ \boldsymbol{u}(t-\xi)),$$

where Φ is the state transition map, $\boldsymbol{u}(t) \in \mathbf{U}_A \subseteq \mathbf{U} \subset \mathbf{R}^l$ the control input at time t, $\mathbf{U}_A$ the admissible input space, and $\mathbf{U}$ the input space. The behavior of the system is monitored via

$$\boldsymbol{y}(t) = \Gamma(t,\ \boldsymbol{u}(t-\xi),\ \boldsymbol{x}(t)),$$

where Γ is the output function, $\boldsymbol{y}(t) \in \mathbf{Y} \subset \mathbf{R}^m$ the output vector at time t and $\mathbf{Y}$ the output space. Let $\mathbf{X}_A$ and $\mathbf{Y}_A$ be the allowed state space and the allowed output space, respectively. Note that, if $\xi = 0$, then $\boldsymbol{u}(t) \in \mathbf{U}_A$ implies $\boldsymbol{x}(t) \in \mathbf{X}_A$ and $\boldsymbol{y}(t) \in \mathbf{Y}_A$. If $0 < \xi < T_s$, the maximum computing time delay that the controlled system can tolerate at time t is defined as the *hard deadline* at that time:

$$d_{\boldsymbol{x}(t)} = \sup_{\boldsymbol{u}(t) \in \mathbf{U}_A} \{\xi : \boldsymbol{x}(t) \in \mathbf{X}_A\}. \tag{8.1}$$

This means that if $\xi > d_{\boldsymbol{x}(t)}$, then the system may move out of the allowed space. Note that, the hard deadline at time t is a function of the state of the controlled system at t. For a control task performed during $[t_0,\ t_1]$, its hard deadline should be the smallest value of $d_{\boldsymbol{x}(t)}$ for all $t \in [t_0,\ t_1]$. For all but very simple cases, it is impossible to get a closed–form relationship between hard deadlines and the allowed space $\mathbf{Y}_A$ (see [SKL85] for more on this). So, Eq. (8.1) is usually used as a

conceptual definition. If the computing time delay associated with a control task is greater than its hard deadline, a *dynamic failure* results [SKL85], meaning that the system has moved out of the allowed state space (for example, the system moved into an unstable region).

8.2.2 How Does the Delay Problem Affect the System Performance ?

As mentioned earlier, system performance and stability are affected by a nonzero computing delay, ξ. We want to investigate how the closed-loop system is affected by this nonzero computing delay. Let a closed-loop system be represented by

$$\dot{\boldsymbol{x}}(t,\xi) = \boldsymbol{f}(t,\ \boldsymbol{x}(t,\xi),\ \xi), \qquad 0 < \xi < T_s. \tag{8.2}$$

Since ξ can usually be made small relative to the control mission lifetime by using parallel processing or high-speed CPUs, Eq. (8.2) can be expanded as a Taylor series, and the subsequent first-order approximation gives

$$\begin{aligned} \dot{\boldsymbol{x}}(t,\xi) &\approx \boldsymbol{f}(t,\ \boldsymbol{x}(t,0),\ 0) + \xi \left(\frac{\partial \boldsymbol{f}(t,\ \boldsymbol{x}(t,0),\ 0)}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}(t,0)}{\partial \xi} + \frac{\partial \boldsymbol{f}(t,\ \boldsymbol{x}(t,0),\ 0)}{\partial \xi} \right) \\ &= \boldsymbol{f}(t,\ \boldsymbol{x}(t,0),\ 0) + \xi\, \boldsymbol{g}(t,\ \boldsymbol{x}), \end{aligned} \tag{8.3}$$

$$\begin{aligned} \text{where } \boldsymbol{g}(t,\ \boldsymbol{x}) &\equiv \frac{\partial \boldsymbol{f}(t,\ \boldsymbol{x}(t,0),\ 0)}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}(t,0)}{\partial \xi} + \frac{\partial \boldsymbol{f}(t,\ \boldsymbol{x}(t,0),\ 0)}{\partial \xi}, \\ \frac{\partial \boldsymbol{f}(t,\ \boldsymbol{x}(t,0),\ 0)}{\partial \xi} &\equiv \frac{\partial \boldsymbol{f}(t,\ \boldsymbol{x}(t,\xi),\ \xi)}{\partial \xi} \Big|_{\xi=0}, \text{ and } \frac{\partial \boldsymbol{x}(t,0)}{\partial \xi} \equiv \frac{\partial \boldsymbol{x}(t,\xi)}{\partial \xi} \Big|_{\xi=0}. \end{aligned}$$

Note that, $\boldsymbol{g}(t,\boldsymbol{x})$ is not a function of ξ. From Eq. (8.3), we conclude that the computing time delay affects the system performance through a permanently acting perturbation.

Since the control system is usually designed under the assumption of $\xi = 0$, the closed-loop system is represented by $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t,\ \boldsymbol{x}(t))$. Suppose there exists a

Lyapunov function $V(t, \boldsymbol{x})$ which is positive definite, decrescent and

$$\dot{V}(t, \boldsymbol{x}) = \frac{\partial V(t, \boldsymbol{x})}{\partial t} + \frac{\partial V(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \boldsymbol{f}(t, \boldsymbol{x}) \leq -r(\| \boldsymbol{x} \|),$$

where $r(\cdot)$ belongs to class K such that the closed–loop system $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t))$ is stable.[2] For $0 < \xi < T_s$, referring to Eq. (8.3), we have

$$\begin{aligned} \dot{V}_\xi(t, \boldsymbol{x}) &= \left(\frac{\partial V(t, \boldsymbol{x})}{\partial t} + \frac{\partial V(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \boldsymbol{f}(t, \boldsymbol{x}(t, 0), 0) \right) + \xi \frac{\partial V(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \boldsymbol{g}(t, \boldsymbol{x}) \\ &\leq -r(\| \boldsymbol{x} \|) + \xi \left| \frac{\partial V(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \boldsymbol{g}(t, \boldsymbol{x}) \right| \\ &\leq -r(\| \boldsymbol{x} \|) + \xi \left\| \frac{\partial V(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \right\| \| \boldsymbol{g}(t, \boldsymbol{x}) \|, \end{aligned}$$

which results in the following two stability conditions

$$\sup_{t \geq t_0, \, \|\boldsymbol{x}\| < \rho} \| \boldsymbol{g}(t, \boldsymbol{x}) \| < \infty, \tag{8.4}$$

$$\sup_{t \geq t_0, \, \|\boldsymbol{x}\| < \rho} \left\| \frac{\partial V(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \right\| < \infty. \tag{8.5}$$

If conditions (8.4) and (8.5) hold, then there exists a $0 < \xi < T_s$ small enough such that $\dot{V}_\xi(t, \boldsymbol{x}) < 0$. Because $V(t, \boldsymbol{x})$ is a positive definite, decrescent function, condition (8.5) is satisfied. Thus, if condition (8.4) is satisfied, then there exists a $0 < \xi < T_s$ small enough such that the closed–loop system is uniformly stable. Note that, though the stability analyses in discrete–time and continuous–time systems are not equivalent, a stable system in continuous time is still stable after the discretization, if the sampling rate is chosen properly. This fact is a basis for the analysis presented in this chapter.

[2] See, for example, *Nonlinear System Analysis* by M. Vidyasagar, Prentice–Hall, 1978 for the definition of class K.

8.2.3 What about Designing a Controller with an Assumed Maximum Value of ξ ?

When designing a controller, one may attempt to handle the computing time delay by using an assumed maximum value of ξ. However, as was discussed in [Woo86], it is impossible to get a precise value of ξ due to the randomness in executing data dependent branches and loops and sharing resources during the execution of control programs. Moreover, in what follows, we will show that this kind of impreciseness in ξ may fail any attempt of using an assumed maximum value. Suppose the controlled plant is described by $G_0(s) = G_1(s)e^{-ds}$, where $d \geq 0$ is the system time delay and not necessarily an integral multiple of the sampling interval. Suppose the computing time delay is $\xi = \xi_0\ T_s,\ 0 < \xi_0 < 1$, then the equivalent controlled plant is approximated by

$$G(s) = G_1(s)\ e^{-Ts}, \qquad T \equiv d + \xi.$$

Let $d = L_0T_s - d_0T_s,\ 0 \leq d_0 < 1$, where $L_0 > 0$ is an integer, then

$$T = L_0T_s + (\xi_0 - d_0)T_s \equiv LT_s - mT_s, \qquad 0 < m < 1,$$

$$\text{where} \quad \begin{cases} L = L_0 + 1 \quad \text{and} \quad m = 1 - (\xi_0 - d_0), & \text{if} \quad \xi_0 > d_0 \\ L = L_0 \qquad \text{and} \quad m = \xi_0 - d_0, & \text{if} \quad \xi_0 \leq d_0. \end{cases}$$

Now, the controlled plant can be represented as

$$G(s) = e^{-LT_ss}\ G_1(s)\ e^{mT_ss}.$$

Let $G(z)$ be the controlled plant $G(s)$ in discrete–time domain. Then by the hold equivalent (zero–order hold), we get

$$G(z) = (1 - z^{-1})\ z^{-L}\ \mathcal{Z}\left\{\frac{G(s)\ e^{mT_ss}}{s}\right\},$$

where $\mathcal{Z}\left\{\frac{G(s)\ e^{mT_s s}}{s}\right\}$ represents the $z-$transform of the time-domain function of $\frac{G(s)\ e^{mT_s s}}{s}$. Therefore, the computing time delay will affect the open-loop zeros, poles at the origin, and the gain. This can be verified by the following simple example.

Example: Suppose $G(s) = \frac{e^{-1.5s}}{1+s}$, $T_s = 1$. By the hold equivalent, the controlled plant in discrete time domain is

$$G_{0.6}(z) = 0.5934\ \frac{z+0.0652}{z^3\ (z-0.3679)}, \quad \text{if} \quad \xi = 0.6, \quad \text{or} \tag{8.6}$$

$$G_{0.3}(z) = 0.1813\ \frac{z+2.4872}{z^2\ (z-0.3679)}, \quad \text{if} \quad \xi = 0.3. \tag{8.7}$$

Obviously, a controller designed for Eq. (8.6) by assuming a maximum computing time delay may not work well for Eq. (8.7).

8.2.4 How Does the Loss Problem Affect System Performance ?

The loss problem is quite different from the delay problem. Let k denote a discrete-time index, and suppose at time k the computer controller fails to update the control output. Because of the D/A converter and the hold circuit, the output of the computer controller does not change when a loss problem occurs; that is, $\boldsymbol{u}(k-1)$ is used over two sampling intervals instead of one sampling interval. For example, loss of one controller's output at time k for the aircraft landing problem in [SKL85] keeps the elevator deflection unchanged, but the aircraft does not remain at the same position. At time $k+1$ the controller computer picks up a new sample $\boldsymbol{y}(k+1)$ and calculates the corresponding control output $\boldsymbol{u}(k+1)$. Thus, loss of one control output is equivalent to the case when the controller computer fails to deliver an output during any one sampling interval over the entire mission lifetime. Let $\Delta\boldsymbol{u}(k) \equiv \boldsymbol{u}(k) - \boldsymbol{u}(k-1)$, then $-\Delta\boldsymbol{u}(k)$ can be treated as a disturbance added

to the system control input at time k. Because this could occur randomly at any time during the mission, the failure to deliver a control output can be treated as a random disturbance to the system. If the computer again fails to deliver a control output at time $k+1$, then the actual control is still the same as $\boldsymbol{u}(k-1)$. Let $\Delta\boldsymbol{u}(k+1) \equiv \boldsymbol{u}(k+1) - \boldsymbol{u}(k-1)$ and suppose $E[\Delta\boldsymbol{u}(k)] = \mathbf{0}$. Since the correlation matrix $E[\Delta\boldsymbol{u}(k)\ \Delta\boldsymbol{u}^T(k+1)]$ may not necessarily be zero, this may be a correlated random disturbance.

For all but simple systems, it is difficult to accurately analyze the effects of computing time delay for the following reasons.

- It is not easy to derive the allowed state space $\mathbf{X}_A$, as was pointed out in [SKL85].
- The computing time delay is a random variable, and its effects may change throughout the entire mission lifetime.
- Different control systems have different structures, thus requiring a separate analysis for each control system. In other words, only case-by-case analyses are possible.

To be more specific, in the next section, the effects of computing time delay on the performance of a typical real-time control system will be analyzed. For qualitative analysis, the generic stability criterion Eq. (8.4) will be used. For quantitative analysis, the example demonstrates how the effects of computing time delay in a control system can be analyzed and evaluated in practice.

8.3 Real–Time Performance Analysis of a Robot Control System

The generic analysis of computing time delay effects can only be solidified with real control systems, because the computing time delay is an application–sensitive measure. In this section we will therefore analyze a robot control system in detail.

To analyze a robot control system under a permanently acting perturbation, we need to know the system dynamic equation, and the control algorithm to be used. The dynamics of a robot arm can be written as

$$\mathbf{H}(\boldsymbol{q})\ \ddot{\boldsymbol{q}} + \mathbf{C}(\boldsymbol{q},\ \dot{\boldsymbol{q}})\ \dot{\boldsymbol{q}} + \mathbf{G}(\boldsymbol{q}) = \boldsymbol{\tau}, \tag{8.8}$$

where $\boldsymbol{q}$ is the vector of joint position, $\mathbf{H}(\boldsymbol{q})$ is the inertia matrix, $\mathbf{C}(\boldsymbol{q},\ \dot{\boldsymbol{q}})\ \boldsymbol{q}$ represents the centrifugal and Coriolis forces, $\mathbf{G}(\boldsymbol{q})$ the vector of gravitational loading, and $\boldsymbol{\tau}$ the vector of torque/force exerted by joint actuators.

8.3.1 Qualitative Analysis

Let the controller be represented by $\boldsymbol{\tau} = \mathbf{c}(\boldsymbol{q},\ \dot{\boldsymbol{q}},\ \xi)$. Note that the controller is originally designed by assuming $\xi = 0$. By letting $\boldsymbol{x}_1 \equiv \boldsymbol{q}$ and $\boldsymbol{x}_2 \equiv \dot{\boldsymbol{q}}$, we can describe the closed–loop system by

$$\begin{aligned} \dot{\boldsymbol{x}}_1 &= \boldsymbol{x}_2 \\ \dot{\boldsymbol{x}}_2 &= \mathbf{H}(\boldsymbol{x}_1)^{-1}\left(-\mathbf{C}(\boldsymbol{x}_1,\ \boldsymbol{x}_2)\ \boldsymbol{x}_2 - \mathbf{G}(\boldsymbol{x}_1) + \mathbf{c}(\boldsymbol{x}_1,\ \boldsymbol{x}_2,\ \xi)\right). \end{aligned} \tag{8.9}$$

These equations can be rewritten as

$$\dot{\boldsymbol{x}}(t,\ \xi) = \boldsymbol{f}(t,\ \boldsymbol{x}(t,\ \xi),\ \xi), \tag{8.10}$$

where $\boldsymbol{x} \equiv \left[\boldsymbol{x}_1^T,\ \boldsymbol{x}_2^T\right]^T$, and

$$\boldsymbol{f}(t,\ \boldsymbol{x}(t,\ \xi),\ \xi) \ \equiv \ \begin{bmatrix} \boldsymbol{x}_2 \\ \mathbf{H}(\boldsymbol{x}_1)^{-1}\left(-\mathbf{C}(\boldsymbol{x}_1,\ \boldsymbol{x}_2)\ \boldsymbol{x}_2 - \mathbf{G}(\boldsymbol{x}_1) + \mathbf{c}(\boldsymbol{x}_1,\ \boldsymbol{x}_2,\ \xi)\right) \end{bmatrix}.$$

For qualitative analysis, referring to Eq. (8.3), we get the standard form of a system with a permanently acting perturbation as

$$\dot{\boldsymbol{x}}(t,\ \xi) \approx \boldsymbol{f}(t,\ \boldsymbol{x}(t,\ 0),\ 0) + \xi\ \boldsymbol{g}(t,\ \boldsymbol{x}).$$

From Eq. (8.10), we get

$$\boldsymbol{g}(t,\boldsymbol{x}) \ = \ \begin{bmatrix} \dfrac{\partial \boldsymbol{x}_2(t,0)}{\partial \xi} \\ \mathbf{a}_{21}\ \dfrac{\partial \boldsymbol{x}_1(t,0)}{\partial \xi} + \mathbf{a}_{22}\ \dfrac{\partial \boldsymbol{x}_2(t,0)}{\partial \xi} + \mathbf{H}(\boldsymbol{x}_1)^{-1}\ \dfrac{\partial \mathbf{c}(\boldsymbol{x}_1,\ \boldsymbol{x}_2,\ 0)}{\partial \xi} \end{bmatrix}$$

where
$$\mathbf{a}_{21} = \frac{\partial}{\partial \boldsymbol{x}_1}\left[\mathbf{H}(\boldsymbol{x}_1)^{-1}\ \left(-\mathbf{C}(\boldsymbol{x}_1,\ \boldsymbol{x}_2)\ \boldsymbol{x}_2 - \mathbf{G}(\boldsymbol{x}_1) + \mathbf{c}(\boldsymbol{x}_1,\ \boldsymbol{x}_2,\ 0)\right)\right]$$
$$\mathbf{a}_{22} = \mathbf{H}(\boldsymbol{x}_1)^{-1}\ \frac{\partial}{\partial \boldsymbol{x}_2}\left(-\mathbf{C}(\boldsymbol{x}_1,\ \boldsymbol{x}_2)\ \boldsymbol{x}_2 + \mathbf{c}(\boldsymbol{x}_1,\ \boldsymbol{x}_2,\ 0)\right).$$

Because each term of $\boldsymbol{g}(t,\boldsymbol{x})$ is bounded above in t, $\sup_{t\geq t_0,\ \|\boldsymbol{x}\|<\rho} \| \boldsymbol{g}(t,\ \boldsymbol{x}) \| < \infty$, that is, the stability condition Eq. (8.4) is satisfied. Thus, we conclude that there exists a $0 < \xi < T_s$ small enough such that the closed–loop system is uniformly stable.

8.3.2 Quantitative Analysis with Respect to System Stability

For any quantitative analysis of the effects of computing time delay, it is necessary to specify the control algorithm to be used. In the robot control system, the controller is $\boldsymbol{\tau} = \mathbf{c}(\boldsymbol{q},\ \dot{\boldsymbol{q}},\ 0)$. If the system parameters in Eq. (8.8) are known, then one can choose the control torque/force vector as

$$\boldsymbol{\tau} \ = \ \mathbf{H}(\boldsymbol{q})\boldsymbol{u}(t) + \mathbf{C}(\boldsymbol{q},\ \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \mathbf{G}(\boldsymbol{q}), \tag{8.11}$$

$$\boldsymbol{u}(t) \ = \ \ddot{\boldsymbol{q}}_d(t) - \mathbf{K}_D\left(\dot{\boldsymbol{q}}(t) - \dot{\boldsymbol{q}}_d(t)\right) - \mathbf{K}_p\left(\boldsymbol{q}(t) - \boldsymbol{q}_d(t)\right), \tag{8.12}$$

where $\boldsymbol{q}_d$ is the desired joint positions and $\mathbf{K}_D$, $\mathbf{K}_p$ are the matrices of controller gains. This is the well–known computed torque algorithm, and on which several

adaptive algorithms (for example, [OS88]) are proposed in the presence of unknown parameters. Taking Laplace transform of Eq. (8.12), we get

$$\mathbf{U}(s) = s^2\mathbf{Q}_d(s) - (\mathbf{K}_D s + \mathbf{K}_p)\Delta(s), \tag{8.13}$$

where $\mathbf{U}(s)$, $\mathbf{Q}_d(s)$ and $\Delta(s)$ are the Laplace transforms of $\boldsymbol{u}(t)$, $\boldsymbol{q}_d(t)$ and $\delta(t) = \boldsymbol{q}(t) - \boldsymbol{q}_d(t)$, respectively. Plugging Eqs. (8.11) and (8.13) into (8.8) and noting that $\mathbf{H}(\boldsymbol{q})$ is nonsingular, we get

$$\left(s^2\,\mathbf{I} + (\mathbf{K}_D s + \mathbf{K}_p)\right)\Delta(s) = \mathbf{0}, \qquad \text{that is,} \quad s^2\,\mathbf{I} + \mathbf{K}_D s + \mathbf{K}_p = \mathbf{0}. \tag{8.14}$$

If there is a nonzero computing time delay, $\delta(t)$ must be replaced by $\delta(t-\xi)$. When the controller is actually implemented on a digital computer, the reference input $\ddot{\boldsymbol{q}}_d(t)$ does not change during one sampling interval, and thus, Eqs. (8.13) and (8.14) become

$$\mathbf{U}(s,\ \xi) = s^2\ \mathbf{Q}_d(s) - (\mathbf{K}_D s + \mathbf{K}_p)\ e^{-s\xi}\ \Delta(s), \qquad \text{and}$$
$$\left(s^2\,\mathbf{I} + (\mathbf{K}_D s + \mathbf{K}_p)\ e^{-s\xi}\right)\Delta(s) = \mathbf{0}.$$

Because $\xi < T_s$ and T_s is small enough to recover the continuous-time signals, we can approximate $e^{-s\xi} \approx 1 - \xi s$ to get

$$\left(s^2\,\mathbf{I} + (\mathbf{K}_D\ s + \mathbf{K}_p)(1 - \xi s)\right)\Delta(s) = \mathbf{0}.$$

$$\text{that is,} \quad s^2\ (\mathbf{I} - \mathbf{K}_D\xi) + s\ (\mathbf{K}_D - \mathbf{K}_p\xi) + \mathbf{K}_p = \mathbf{0}. \tag{8.15}$$

We may choose $\mathbf{K}_D = diag\,[K_{Di}]$, $\mathbf{K}_p = diag\,[K_{pi}]$, $K_{Di},\ K_{pi} > 0$, $i = 1,\ 2,\ \cdots,\ n$, such that the closed-loop system becomes uncoupled, linear, and exponentially stable. Then, Eq. (8.15) becomes

$$s^2\ (1 - K_{Di}\xi) + s\ (K_{Di} - K_{pi}\xi) + K_{pi} = 0, \qquad i = 1,\ 2,\ \cdots,\ n. \tag{8.16}$$

We can derive from Eq. (8.16) the least upper bound of ξ for system stability:

$$\xi < \inf_i \left(\min \left\{ \frac{K_{Di}}{K_{pi}}, \frac{1}{K_{Di}} \right\} \right), \quad i = 1, 2, \cdots, n. \tag{8.17}$$

This upper bound of ξ can be viewed as a hard deadline with respect to system stability and used as an extra constraint on the selection of controller gains.

8.3.3 Quantitative Analysis with Respect to System Performance

In practice, we need not only to know the least upper bound of ξ with respect to system stability, but also the quantitative performance changes caused by it. As stated before, the controller is designed under the assumption of $\xi = 0$. If the controller gains are diagonal matrices, then Eq. (8.14) becomes

$$s^2 + K_{Di}\, s + K_{pi} = 0, \quad i = 1, 2, \cdots, n. \tag{8.18}$$

Comparing this with the standard form of a second order system $s^2 + 2\,\zeta_i\,\omega_{ni}\,s + \omega_{ni}^2 = 0$, we choose

$$K_{Di} = 2\,\zeta_i\,\omega_{ni}, \quad K_{pi} = \omega_{ni}^2, \quad i = 1, 2, \cdots, n, \tag{8.19}$$

where ζ_i and ω_{ni} are the closed-loop damping ratio and the natural frequency of subsystem i, respectively.

We consider the relative displacement of the closed-loop system's poles from their originally-designed positions as a result of the nonzero computing time delay. Let s_a be the actual pole position which is moved from its desired position, s_d, due to the presence of $\xi > 0$. s_d and s_a are given by Eqs. (8.18) and (8.16), respectively, as

$$\begin{aligned} s_d &= -\frac{1}{2}\,K_{Di} \pm j\,\frac{1}{2}\,\sqrt{4\,K_{pi} - K_{Di}^2}, \quad \text{and} \\ s_a &= \frac{-(K_{Di} - K_{pi}\,\xi) \pm j\,\sqrt{4\,(1 - K_{Di}\,\xi)\,K_{pi} - (K_{Di} - K_{pi}\,\xi)^2}}{2\,(1 - K_{Di}\,\xi)}. \end{aligned}$$

We define the maximum relative displacement of poles as the performance tolerance

$$\begin{aligned} \alpha &\equiv \max_i \frac{|\sigma_{di} - \sigma_{ai}|}{\sigma_{di}}, \qquad (8.20) \\ \text{where} \quad \sigma_{di} &= \| s_d \|_2 = \sqrt{K_{pi}}, \quad \text{and} \\ \sigma_{ai} &= \| s_a \|_2 = \sqrt{\frac{K_{pi}}{1 - K_{Di}\,\xi}}, \quad \sigma_{ai} \geq \sigma_{di}. \end{aligned}$$

Therefore, from Eqs. (8.20) and (8.19), if the performance tolerance α is specified, we get

$$\xi < \begin{cases} \max_i \left(\dfrac{1}{2\,\zeta_i\,\omega_{ni}} \left| 1 - \dfrac{1}{(1-\alpha)^2} \right| \right) & \text{if} \quad \alpha \neq 1, \\ \max_i \left(\dfrac{3}{8\,\zeta_i\,\omega_{ni}} \right) & \text{if} \quad \alpha = 1, \end{cases} \qquad (8.21)$$

for $i = 1,\ 2,\ \cdots,\ n$. This is the upper bound of ξ with respect to the performance tolerance for the computed torque algorithm. That is, the computing time delay must satisfy Eq. (8.21) in order not to violate the specified performance tolerance. This upper bound can be viewed as a hard deadline with respect to system performance, and used as an index to select a CPU to implement the control algorithm.

The computed torque algorithm Eq. (8.11) is based on the assumption that the parameters of Eq. (8.8) are completely known. However, they are, in fact, unknown, or not known precisely. Some adaptive schemes may be used to estimate these parameters and then the computed torque algorithm or variations thereof [OS88]. If it is assumed that the estimators give the true values and a perfect tracking of time–varying parameters, then performance changes can be analyzed separately while figuring the time consumed by the estimators in the computing time delay. This implies that Eqs.(8.17) and (8.21) can also be used as an approximate analysis for adaptive control methods which are based on the computed torque algorithm. Since the parameter estimators and the adaptive algorithm require a longer computing time delay than non–adaptive ones, it is important to analyze the effects of computing time

delay. To this end, in the next two sections, we will analyze the effects of computing time delay for both a knowledge–based coordinator and an NN–based coordinator for two robots holding an object. Unlike the single–robot control system treated in the previous section, those examples deal with a complex control system for which the system dynamic equations are either unknown or known only qualitatively.

8.4 Real–Time Performance Analysis of the Knowledge–Based Coordination System

In a knowledge–based or an NN–based control/coordination system, the system dynamic equations are usually unknown or known only qualitatively, and one cannot describe the control actions mathematically. This implies that evaluation of the effects of computing time delay effects on such systems may depend heavily on simulations and/or experiments. Moreover, as was concluded in previous sections, loss of control outputs is equivalent to introduction of random disturbances. However, disturbance– and/or noise–rejection ability of a knowledge–based or an NN–based control/coordination system has not been addressed in most relevant literature. Therefore, we want to analyze the effects of both delay and loss problems for a knowledge–based and an NN–based control/coordination system. In this section, the effects of computing time delay on system performance is analyzed for a knowledge–based coordination system developed in Chapter 5. The NN–based coordinator developed in Chapter 7 will be investigated in next section.

The problem is to coordinate two 2–link robots holding a rigid object. The solution consists of two levels: the high level is a knowledge–based coordinator (KBC) and the low–level subsystems are two robots each with a separately designed servo controller. The basic configuration of this example is shown in Fig. 5.5. The Carte-

sian frame is fixed at the base of robot 1, and the trajectories of the object and the robots' end–effectors are specified relative to this frame. We want to move the object forward and then backward in X direction while keeping the height in Y direction constant. The detail of the problem statement and the principle of the KBC are presented in Chapter 5.

In the example of two 2–link robots holding an object, each robot is position controlled with the computed torque algorithm. For such a control algorithm, an upper bound of computing time delay with respect to system stability has been derived in Eq. (8.17), and rewritten below as

$$\xi < \inf_i \left(\min \left\{ \frac{K_{Di}}{K_{pi}}, \frac{1}{K_{Di}} \right\} \right), \quad i = 1, 2. \tag{8.22}$$

For the position controller with the computed torque algorithm, obviously there are many choices of the controller gains to satisfy different performance requirements in addition to the conditions of system stability. However, the selection of the controller gains must satisfy the condition of Eq. (8.22) concerning the computing time delay. In this coordination scheme, the KBC forms a high–level coordinator, and the internal structure and/or parameters of the low–level subsystems are not affected. Therefore, this upper bound can be used to approximately analyze the effects of computing time delay on the stability of multiple–system with the KBC.

The search process of the KBC is determined by the decision tree shown in Fig. 3.2, and the modification to the system reference input is computed with Eq. (5.3). Then the average search speed is given by setting $K = 0.5$. After the n-th iteration, the magnitude of modification to the reference input is

$$|c_j^n| = \frac{|a_0^0 - b_0^0|}{2^{n+1}}.$$

In the worst case, the search will not be completed until

$$|c_j^{n-1}| - |c_l^n| \leq \delta \quad \text{or} \quad \frac{|a_0^0 - b_0^0|}{2^{n+1}} \leq \delta,$$

where $\delta > 0$ is a pre–specified tolerance. Usually, the initial value is $c_0^0 = 0$. Therefore, the maximum number of iterations with the average search speed is given by

$$n = \left\lceil \frac{\ln |a_0^0 - b_0^0| - \ln \delta}{\ln 2} - 1 \right\rceil. \tag{8.23}$$

For example, if $a_0^0 = -5$, $b_0^0 = 5$, $c_0^0 = 0$, $K = 0.5$ and $\delta = 0.001$, then in the worst case, the KBC needs 13 iterations ($n = 13$) to complete the search process. From Eq. (8.23), the maximum computing time may be estimated, which is then used to select the controller gains and satisfy the condition Eq. (8.22).

Let the cost function of two 2–link robots holding an object be defined by

$$J = \frac{1}{N} \sum_{k=1}^{N} \left(\boldsymbol{E}_I^T(k)\ \boldsymbol{E}_I(k) + \boldsymbol{E}_e^T(k)\ \boldsymbol{E}_e(k) \right), \tag{8.24}$$

where $\boldsymbol{E}_I(k) = \frac{1}{2}(\boldsymbol{f}_1(k) - \boldsymbol{f}_2(k))$ and $\boldsymbol{E}_e(k) = (\boldsymbol{f}_{1d}(k) - \boldsymbol{f}_1(k)) + (\boldsymbol{f}_{2d}(k) - \boldsymbol{f}_2(k))$ are the internal force and external force error, respectively.

According to the robot dynamic parameters, three groups of controller gains are designed and listed in Table 8.1. The sampling interval is $T_s = 10\ ms$. For the first group $K_{pi} = 157.90$, $K_{Di} = 18.85$, the upper bound of the computing time delay is 53.0 ms. This implies that the system can tolerate a delay of $5T_s$. The cost functions corresponding to different delays are plotted in Fig. 8.2. The system becomes unstable if $6T_s$ extra delays are introduced. So, in this case, the delay problem will never affect the system stability. Fig. 8.3 shows the cost functions using the second group of controller gains $K_{pi} = 986.96$, $K_{Di} = 47.12$ for different delays. The upper bound of the computing time delay is 21.2 ms. In our simulation, the system was shown to become unstable if two sampling–interval delays are introduced.

Controller gains		Closed–loop system		Upper bound of ξ
K_{pi}	K_{Di}	damping ratio ζ	natural frequency ω_n	(ms)
157.90	18.85	0.75	$2 \times 2\pi(rad/s)$	53.0
986.96	47.12	0.75	$5 \times 2\pi(rad/s)$	21.2
3947.84	94.25	0.75	$10 \times 2\pi(rad/s)$	10.6

Table 8.1: Tested controller gains and the upper bounds of computing time delay.

If the third group of controller gains ($K_{pi} = 3947.84, K_{Di} = 94.25$) are used, then the upper bound of the computing time delay is reduced to 10.6 ms. Because the sampling interval is $T_s = 10$ ms, the computing time delay will greatly affect the system performance. The cost functions corresponding to different computing time delays are plotted in Fig. 8.4. The system becomes unstable at $\xi = 7ms$. Obviously, this group of controller gains cannot be used due to the potential problem caused by the computing time delay. Therefore, we conclude that the selection of controller gains should not only satisfy the desired system responses but also the upper bound of computing time delay — Eq. (8.22). From Figs. 8.2 - 8.4, we also conclude that the system performance is improved by adding the KBC for different computing time delays. Note that the KBC is designed only to improve the system performance (more precisely, to reduce the internal force); we cannot rely on the KBC if the original system becomes unstable.

The loss problem is also simulated for the coordination of two 2–link robots holding an object. In the simulation, ten consecutive outputs are lost after the object reached the maximum or the minimum velocity. The controller gains are $K_{pi} = 157.90$ and $K_{Di} = 18.85$. The simulation results are summarized in Table 8.2.

	Cost function	
	without the KBC	with the KBC
no output loss	92.584	13.228
lose 10 outputs at the point of maximum velocity	92.670	13.224
lose 10 outputs at the point of minimum velocity	92.678	13.218

Table 8.2: The cost functions with different output losses for the KBC.

These results indicate that the KBC is not sensitive to the loss problem, agreeing with the observation in [SC88]: PID–type controllers are not sensitive to the loss problem.

8.5 Real–Time Performance Analysis of the NN–Based Coordination System

One of the advantages of NNs is that parallel processing can be easily implemented. By distributing the computational burden to all nodes, the effects of the delay problem on system performance can be reduced. However, as a controller or coordinator, the numbers of the nodes at each layer may not be as large as those in such applications as pattern recognition and image processing. So, the NN–based controller or coordinator may be implemented in software, and thus the computing time delay may become significant, as shown in the example below. Moreover, it is also true that the loss problem may affect the system performance as discussed in Section 8.1. In what follows, the NNBC developed in Chapter 7 is evaluated for both the delay and the loss problems.

Computation	Multiplication (M)	Addition (A)	Division (D)	Exponential (E)
HIDDEN layer	$N_1\ N$	$N_1\ (N+1)$	N_1	N_1
OUTPUT layer	$N_2\ N_1$	$N_2\ (N_1+1)$	N_2	N_2
weights from HIDDEN to OUTPUT layer	$4\ N_1\ N_2$	$3\ N_1\ N_2$	0	0
weights from INPUT to HIDDEN layer	$(4+N_2)\ N\ N_1$	$(1+N_2)\ N\ N_1$	0	0
thresholds	N_1+N_2	N_1+N_2	0	0
total	$M = 5\ N\ N_1 + 5\ N_1\ N_2 + N\ N_1\ N_2 + N_1 + N_2$ $A = 2\ N\ N_1 + 2\ N_1 + 2\ N_2 + 4\ N_1\ N_2 + N\ N_1\ N_2$ $D = N_1 + N_2$ and $E = N_1 + N_2$			

Table 8.3: The computational requirement of a three–layer perceptron.

We must consider the computational requirement of a three–layer perceptron with the standard BP algorithm in order to obtain quantitative knowledge about the computational requirement. The details of the standard BP algorithm can be found, for example, in [RM86], and thus omitted here. Suppose there are N input nodes, N_1 hidden nodes, and N_2 output nodes, and each node performs scalar operations only. The computational requirement is given in Table 8.3. For example, when $N = 4$, $N_1 = 15$, and $N_2 = 1$, the computation required is 451 multiplicative, 272 additive, 16 divisional, and 16 exponential operations. In the NNBC proposed in

Chapter 7, each node has the ability to perform vector operations. For the example of coordinating two 2–link robots holding an object, the dimension of the vector is four. This implies that the computational requirement is at least four times the above figure for the neural network part alone. Therefore, the computational burden is significant if the neural network is implemented in software.

Similar to the evaluation of the KBC, using the same cost function Eq. (8.24), three groups of controller gains in Table 8.1 are evaluated. Fig. 8.5 shows the results using the first group controller gains: $K_{pi} = 157.90$, $K_{Di} = 18.85$. Without the NNBC, the upper bound is still 53 ms. However, compared with Fig. 8.2, we conclude that the NNBC is more prone to be unstable than the KBC as a result of computing time delay. Note that the unstability in Figs. 8.2 and 8.5 will never occur, since the computing time delay always less than one sampling interval. The cost functions for the second and third groups of controller gains are plotted in Figs. 8.6 and 8.7. The performance of the NNBC is not degraded due to the computing time delays as long as the two–robot system is still stable in the presence of the computing time delay. Similar to the KBC, the NNBC is used to improve the system performance, but not to stabilize an unstable multiple–system.

The NNBC is also tested for the loss problem. The simulation arrangement is the same as that for KBC, that is, ten control outputs are lost after the object reached the maximum or the minimum velocity. The controller gains are $K_{pi} = 157.90$ and $K_{Di} = 18.85$, and the results are presented in Table 8.4. These results indicate that the NNBC is not sensitive to the loss problem either. This implies that the weights of the NN for this example may not necessarily have to be updated in every sampling interval.

	Cost function	
	without the NNBC	with the NNBC
no output loss	92.584	0.641
lose 10 outputs at the point of maximum velocity	92.670	0.652
lose 10 outputs at the point of minimum velocity	92.678	0.659

Table 8.4: The cost function with different output losses for the NNBC.

8.6 Summary

The increasing use of digital computers to implement real–time controllers has made it essential to carefully study the effects of computing time delay on the stability and performance of controlled systems. This computing time delay is *different* from the usual system time delay; it is a random delay resulting from the execution of control programs on a digital computer.

The effects of computing time delay on control system performance are classified into delay and loss problems, which are then analyzed for both general and special cases. A generic criterion is derived for the qualitative analysis of the delay problem. Since any quantitative analysis requires the detailed knowledge of the controlled system and the control algorithm to be used, we have chosen a prototypical, real–time control system with a commonly–used control algorithm — a robot control system with the computed torque algorithms — to give a detailed account of computing time delay effects. For such a system, the upper bounds of computing time delay for system stability and performance are derived as an extra constraint on controller

design and selection of CPUs to implement the control algorithm.

Both the knowledge–based and the NN–based coordinator are evaluated for the delay and loss problem via simulation. Since the internal structure and/or parameters of the low–level subsystems are not affected by adding the coordinators, the servo controllers of the subsystems are designed separately from, and independently of, the others. Therefore, the upper bounds of computing time delay are still valid even when two such subsystems cooperate. Moreover, the maximum number of iterations with an average search speed is derived and can be used to estimate the computation time required for the KBC. Both the qualitative and quantitative analyses of the robot control/coordination system have demonstrated how system performance is affected by the computing time delay and how a given system can be evaluated based on the computing time delay.

By adding the coordinator, the system performance is improved even under the effect of computing time delay, as long as the coordinated subsystems are stable in the presence of computing time delay. The coordinator cannot be used to stabilize an unstable multiple–system. Moreover, for coordinating two 2–link robots holding an object, both KBC and NNBC are shown to be not sensitive to the loss problem.

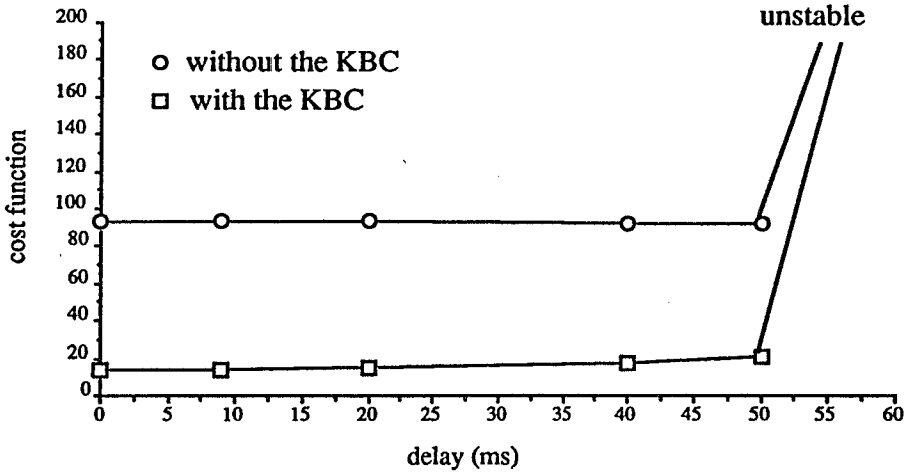


Figure 8.2: Performance of the KBC with the 1st group of gains.

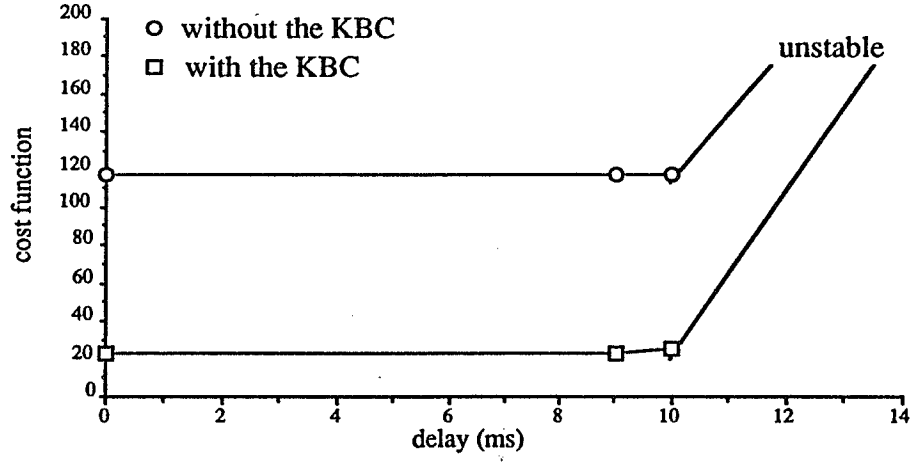


Figure 8.3: Performance of the KBC with the 2nd group of gains.

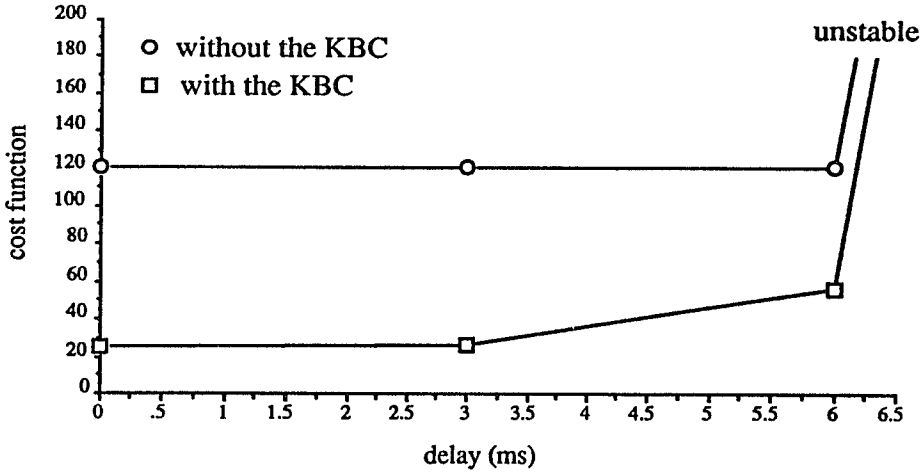


Figure 8.4: Performance of the KBC with the 3rd group of gains.

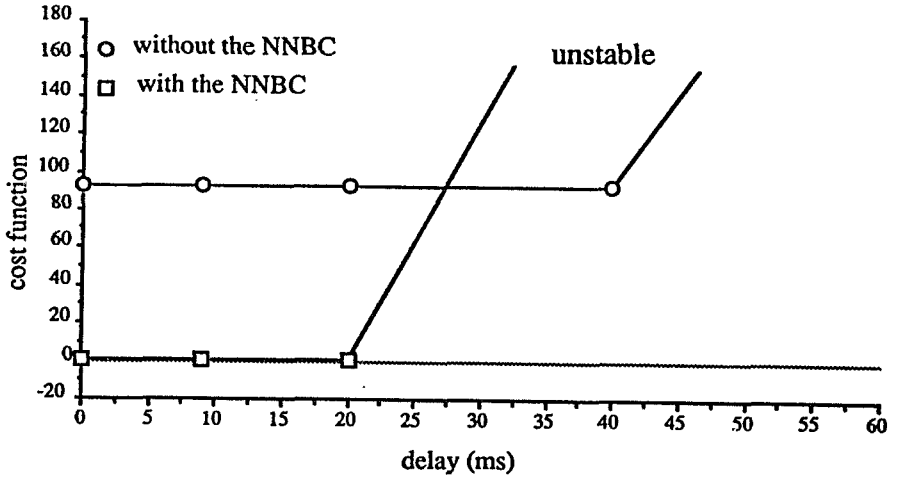


Figure 8.5: Performance of the NNBC with the 1st group of gains.

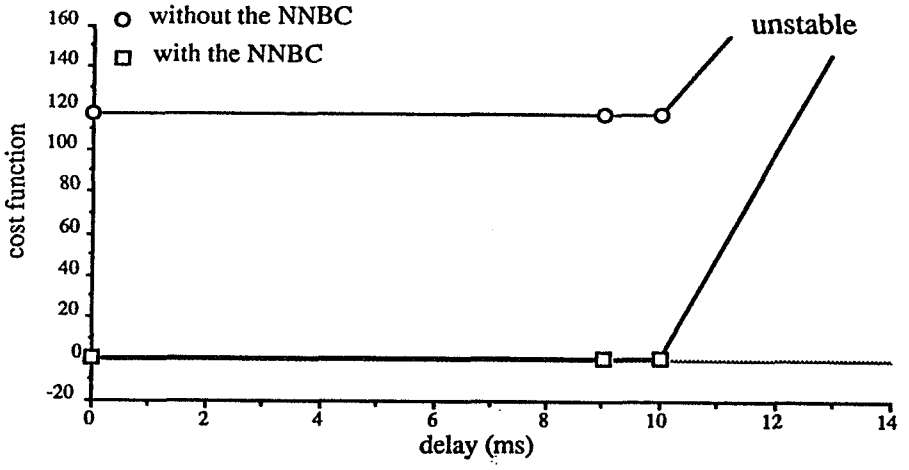


Figure 8.6: Performance of the NNBC with the 2nd group of gains.

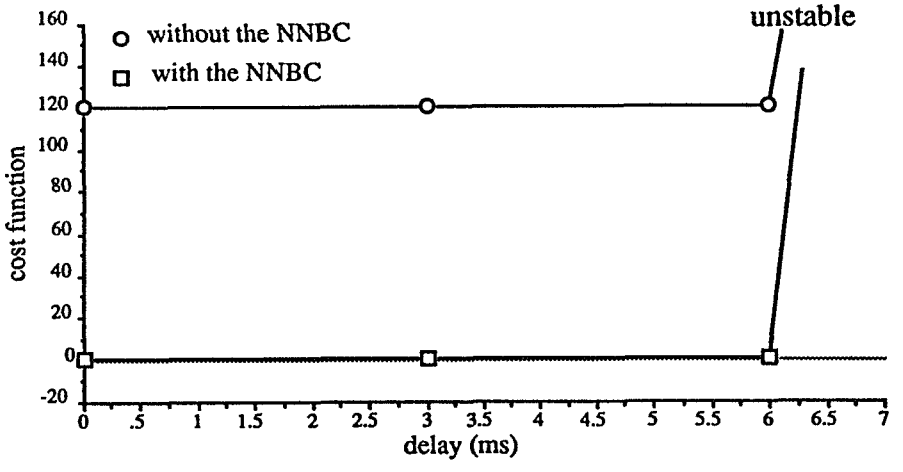


Figure 8.7: Performance of the NNBC with the 3rd group of gains.

CHAPTER IX

CONCLUSION

Focusing on the coordination of multiple systems, two practical and general designs have been developed: a knowledge–based coordinator (KBC) and an NN–based coordinator (NNBC). The basic principles we followed are (1) in a hierarchical structure; the higher the level is the more intelligence it has for decision making, but the less precision with which it knows about the internal structure and parameters of lower levels, and (2) different levels should be independent of each other in the sense that the internal structure and parameters are not affected by adding other levels. With either of the proposed coordinators, the coordinated system forms a hierarchical structure in which the high level is the coordinator and the low level is the coordinated subsystems. By adding either of the coordinators, the two principles are satisfied. This implies that some commercially–designed servo controllers may be directly coordinated for some tasks which require multiple–system cooperation. Considering the implementation of a control algorithm on digital computers, the effects of computing time delay has also been analyzed in this dissertation. For a robot control system, the upper bounds of computing time delay are derived as extra constraints on the control system design. Both the KBC and the NNBC are evaluated in terms of computing time delay.

The KBC combines the techniques of intelligent control and neural networks. The low–level subsystems are viewed as a mapping from the system reference inputs to the system outputs. In order not to interfere in the internal structure and/or parameters of the low–level subsystems, the only action that the coordinator takes is to issue a sequence of appropriate reference inputs. The basic idea is to estimate the effects of the reference inputs using a predictor and to modify them through a search in a knowledge base in order to achieve the desired performance. A general–purpose, MIMO predictor has been designed using neural networks. By introducing the predictor, the knowledge base for multiple–system coordination is greatly simplified, and each reference input is evaluated before its actual application. The NN–based predictor deals with the unknown parameters and/or time–varying properties of the coordinated multiple–system, while the KBC emphasizes logical reasoning and decision making.

The NNBC is designed based on the property that a multilayer perceptron can be used to approximate any continuous mapping. The basic structure of the NNBC is a multilayer perceptron. The outputs of the NNBC are the coordination commands to the low–level subsystems, that is, the outputs of the NN are the control variables of the low–level subsystems. NNs are usually trained by using the output errors of the network. However, unlike the indirect control schemes such as those proposed in [NP90], when an NN is used to control a plant directly, the output errors of the network are unknown, since the desired control actions are unknown. Therefore, in designing the NNBC, one of the key problems is to develop an efficient training algorithm. We want to train the NNBC by using the output errors of the controlled plant, instead of using the unknown output error of the NN. A simple training algorithm has been developed which enables the NN to be trained by the output

errors of the controlled plant. In this way, the proposed algorithm enhances the NN's ability to handle control applications.

The effects of computing time delay on the stability and performance of controlled systems have been carefully studied for both a general robot control system and the coordinators developed in this dissertation. The effects of the computing time delay are classified into delay and loss problems. We concluded that the delay problem affects the control system through a permanently acting perturbation, while the loss problem is equivalent to random disturbances. For qualitative analysis of the delay problem, a generic criterion is derived in terms of system stability. For quantitative analysis, upper bounds of computing time delay on system stability and performance are derived for a robot control system with the computed torque algorithm. These upper bounds can be used as extra constraints on controller design and CPU selection to implement the control algorithm. Both the KBC and NNBC are evaluated in terms of the delay problem and the loss problem via simulation. Since the internal structure and/or parameters of the low–level subsystems are not affected by adding the coordinators, the upper bounds of computing time delay still hold when two such subsystems work cooperately. This is the basis of evaluation for both the KBC and NNBC.

The main contributions of this dissertation are summarized as follows:

1. A KBC has been developed by combining the techniques of intelligent control and neural networks. Using an NN–based predictor, the design of the knowledge base for multiple–system coordination is simplified. The KBC emphasizes logical reasoning, while the ability of learning mainly relies on the NN.

2. A direct adaptive control and coordination scheme is developed using neural networks. A simple training algorithm is proposed based on the BP algorithm.

This algorithm solved a major problem of neural networks in some control applications in which the output errors of the network are unavailable for on–line training.

3. As a high–level coordinator, by adding either of the KBC or NNBC the coordination system forms a hierarchical structure. Therefore, some industrial robots with independently and commercially designed servo controllers could be coordinated to perform more sophisticated tasks than originally intended.

4. On–line collision avoidance of multiple robots working in a common workspace can be accomplished by a simple method, without imposing any constraint on path planning, trajectory planning, and design of servo controllers.

5. By using either the KBC or the NNBC, we open a new way to solve the problem of coordinating multiple robots holding an object.

6. Replacing a conventional controller, the NN–based controller overcomes the negative effects of a long system response delay, nonlinear elements with dead zone and/or saturation, and process noises in a class of industrial process control systems.

7. The NN–based predictor alone can be used as a general–purpose predictor for many industrial applications.

8. The theory of performance evaluation on real–time computer control systems is extended. The computing time delay problem and the control output loss problem are defined, and their effects on system performance are analyzed in detail. For a certain type of robot control systems, the upper bounds of the computing time delay are derived which can be used as an extra constraint on controller design and CPU selection to implement a control algorithm.

The future work of this research includes:

1. Extend the KBC to coordinate more than two robots in a common workspace to avoid collision.
2. Investigate the application of the general–purpose predictor in intelligent decision support systems.
3. Develop a method to determine the number of hidden nodes when designing a NN–based controller.
4. Extend the NNBC to coordinate two multiple–link robots.

APPENDIX

APPENDIX A

SIMULATION OF TWO 2–LINK ROBOTS HOLDING AN OBJECT

A1. Kinematics and Dynamics of a 2–Link Robot

Direct Kinematics

Referring to Fig. 3.6, when the joint position $\boldsymbol{q} \equiv [q_1,\ q_2]^T$ is specified, the position of the end–effector is given by

$$P_x = L_1\ c_1 + L_2\ c_{12}$$

$$P_y = L_1\ s_1 + L_2\ s_{12},$$

where L_i is the length of link i,

$$c_i \equiv \cos q_i, \qquad s_i \equiv \sin q_i, \qquad i = 1,\ 2, \quad \text{and}$$

$$c_{12} \equiv \cos(q_1 + q_2), \qquad s_{12} \equiv \sin(q_1 + q_2).$$

Inverse Kinematics

Given the position of the end–effector $(P_x,\ P_y)$, the joint positions are computed as follows:

$$\text{(1). compute} \quad \theta = \arctan \frac{P_y}{P_x},$$

$$\text{(2). compute} \quad \theta_1 = \arccos \left(- \frac{L_2^2 - L_1^2 - (P_x^2 + P_y^2)}{2\ L_1\ \sqrt{P_x^2 + P_y^2}} \right),$$

(3). compute $q_1 = \theta - \theta_1$,

(4). compute $q_2 = \arccos\left(\dfrac{P_x^2 + P_y^2 - L_1^2 - L_2^2}{2\ L_1\ L_2}\right)$.

Jacobian Matrix

$$\mathbf{J} \equiv \begin{bmatrix} \dfrac{\partial P_x}{\partial q_1} & \dfrac{\partial P_x}{\partial q_2} \\ \dfrac{\partial P_y}{\partial q_1} & \dfrac{\partial P_y}{\partial q_2} \end{bmatrix} = \begin{bmatrix} -(L_1\ s_1 + L_2\ s_{12}) & -L_2\ s_{12} \\ L_1\ c_1 + L_2\ c_{12} & L_2\ c_{12} \end{bmatrix},$$

$$\mathbf{J}^{-1} = \begin{bmatrix} L_2\ c_{12} & L_2\ s_{12} \\ -(L_1\ c_1 + L_2\ c_{12}) & -(L_1\ s_1 + L_2\ s_{12}) \end{bmatrix} \frac{1}{L_1\ L_2\ s_2},$$

$$\dot{\mathbf{J}} = \begin{bmatrix} -L_1\ c_1\ \dot{q}_1 - L_2\ c_{12}\ (\dot{q}_1 + \dot{q}_2) & -L_2\ c_{12}\ (\dot{q}_1 + \dot{q}_2) \\ L_1\ s_1\ \dot{q}_1 - L_2\ s_{12}\ (\dot{q}_1 + \dot{q}_2) & -L_2\ s_{12}\ (\dot{q}_1 + \dot{q}_2) \end{bmatrix}.$$

Dynamics

The dynamic equation of a robotic manipulator in joint space, in general form, is

$$\mathbf{H}(\boldsymbol{q})\ \ddot{\boldsymbol{q}} + \mathbf{C}(\boldsymbol{q},\ \dot{\boldsymbol{q}})\ \dot{\boldsymbol{q}} + \mathbf{G}(\boldsymbol{q}) + \mathbf{J}^T\ \boldsymbol{f} = \boldsymbol{\tau}, \tag{A.1}$$

where $\boldsymbol{q}$, $\boldsymbol{\tau}$, $\boldsymbol{f}$ are the joint position, joint torque, and the force exerted on the end-effector, respectively. $\mathbf{H}(\boldsymbol{q})$ is the inertia matrix, $\mathbf{C}(\boldsymbol{q},\ \dot{\boldsymbol{q}})\ \dot{\boldsymbol{q}}$ represents the Coriolis and centrifugal forces, $\mathbf{G}(\boldsymbol{q})$ represents the gravitational force, and $\mathbf{J}$ is the Jacobian matrix. For a 2-link robot, we have $\boldsymbol{q} \equiv [q_1,\ q_2]^T$, $\boldsymbol{\tau} \equiv [\tau_1,\ \tau_2]^T$, and $\boldsymbol{f} \equiv [f_x,\ f_y]^T$. Then each term of Eq. (A.1) is given as

$$\mathbf{H}(\boldsymbol{q}) = \begin{bmatrix} m_1 L_{c1}^2 + m_2(L_1^2 + L_{c2}^2 + 2L_1 L_{c2} c_2) + I_1 + I_2, & m_2 L_1 L_{c2} c_2 + m_2 L_{c2}^2 + I_2 \\ m_2\ L_1\ L_{c2}\ c_2 + m_2\ L_{c2}^2 + I_2, & m_2\ L_{c2}^2 + I_2 \end{bmatrix}$$

$$\mathbf{C}(\boldsymbol{q},\ \dot{\boldsymbol{q}}) = \begin{bmatrix} -2\ m_2\ L_1\ L_{c2}\ s_2\ \dot{q}_2, & -m_2\ L_1\ L_{c2}\ s_2\ \dot{q}_2 \\ m_2\ L_1\ L_{c2}\ s_2\ \dot{q}_1, & 0 \end{bmatrix}$$

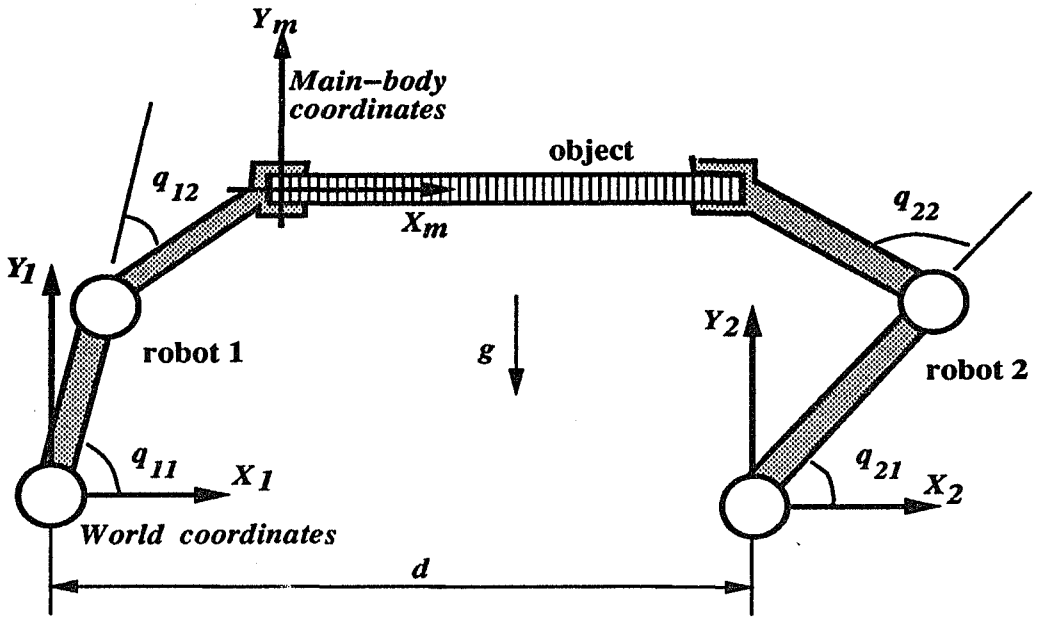


Figure A.1: Basic configuration of the simulated system.

$$\mathbf{G}(q) = \begin{bmatrix} m_1 \, L_{c1} \, g \, c_1 + m_2 \, g \, (L_{c2} \, c_{12} + L_1 \, c_1) \\ m_2 \, L_{c2} \, g \, c_1 2 \end{bmatrix}.$$

m_i, L_{ci} and I_i are the mass, mass center and moment of inertial of link i, respectively. g is the gravitational acceleration.

A2. Simulation of Two 2–Link Robots Holding an Object

In the mechanism of two 2–link robots holding an object, it is assumed that there is no relative motion between the end–effectors and the object. The object is called the main body of the mechanism because the position of every link in the mechanism can be determined by given the position of the main body [Wal86]. The basic configuration is shown in Fig. A.1. Let $\boldsymbol{v} = \dot{\boldsymbol{P}}$ be the velocity of the object, then the motion of the object is described by

$$m \, \dot{\boldsymbol{v}} - \boldsymbol{f}_1 - \boldsymbol{f}_2 = \begin{bmatrix} 0 \\ -m \, g \end{bmatrix},$$

where $\boldsymbol{f}_i$ is the force exerted by robot i, $i = 1,\ 2$. m is the mass of the object and g is the gravitational acceleration. Referring to Eq. (A.1), the force constrained dynamic equation of robot i in joint space is

$$\mathbf{H}_i(\boldsymbol{q}_i)\ \ddot{\boldsymbol{q}}_i + \mathbf{C}_i(\boldsymbol{q}_i,\ \dot{\boldsymbol{q}}_i)\ \dot{\boldsymbol{q}}_i + \mathbf{G}_i(\boldsymbol{q}_i) + \mathbf{J}_i^T\ \boldsymbol{f}_i = \boldsymbol{\tau}_i$$

Then the simulation procedures are listed below.

1. Given: the acceleration of the object $\dot{\boldsymbol{v}}$.

   Compute: the velocity of the object $\boldsymbol{v}$ by integration.

2. Given: the velocity of the object $\boldsymbol{v}$.

   Compute: the position of the object $\boldsymbol{P}$ by integration.

3. Given: the position of the object $\boldsymbol{P}$.

   Compute: the positions of the end-effectors

   $$\text{end-effector 1:}\quad \boldsymbol{P}_1 = \boldsymbol{P},$$

   $$\text{end-effector 2:}\quad \boldsymbol{P}_2 = \boldsymbol{P} - [\mathit{base\_distance},\ 0]^T.$$

4. Given: the position of the end-effectors $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$.

   Compute: the joint positions of the two robots $\boldsymbol{q}_1 \equiv [q_{11},\ q_{12}]^T$ and $\boldsymbol{q}_2 \equiv [q_{21},\ q_{22}]^T$ by inverse kinematics.

5. Given: the joint accelerations $\ddot{\boldsymbol{q}}_1$ and $\ddot{\boldsymbol{q}}_2$.

   Compute: the joint velocities $\dot{\boldsymbol{q}}_1$ and $\dot{\boldsymbol{q}}_2$ by integration.

6. Given: the joint velocity $\dot{\boldsymbol{q}}_i$ and position $\boldsymbol{q}_i$, $i = 1,\ 2$.

   Compute: the robots' dynamic parameters $\mathbf{H}_i(\boldsymbol{q}_i)$, $\mathbf{C}_i(\boldsymbol{q}_i,\ \dot{\boldsymbol{q}}_i)$, $\mathbf{G}_i(\boldsymbol{q}_i)$, Jacobian matrices $\mathbf{J}_i$ and its derivative $\dot{\mathbf{J}}_i$, $i = 1,\ 2$.

7. Given: the joint velocity $\dot{q}_i$, position q_i and joint torque τ_i, $i = 1, 2$.

   Compute:

$$K_i = \tau_i - \mathrm{C}_i\, \dot{q}_i - \mathrm{G}_i, \qquad r_i = -\dot{\mathrm{J}}_i\, \dot{q}_i, \qquad i = 1, 2,$$

   and form the equation $\mathrm{A}X = B$, where

$$\mathrm{A} = \begin{bmatrix} \mathrm{H}_0 & 0 & 0 & -\mathrm{I}_2 & -\mathrm{I}_2 \\ 0 & \mathrm{H}_1 & 0 & \mathrm{J}_1^T & 0 \\ 0 & 0 & \mathrm{H}_2 & 0 & \mathrm{J}_2^T \\ -\mathrm{I}_2 & \mathrm{J}_1 & 0 & 0 & 0 \\ -\mathrm{I}_2 & 0 & \mathrm{J}_2 & 0 & 0 \end{bmatrix}, \qquad X = \begin{bmatrix} \dot{v} \\ \ddot{q}_1 \\ \ddot{q}_2 \\ f_1 \\ f_2 \end{bmatrix},$$

$$B = \begin{bmatrix} [0,\ -m\ g]^T \\ K_1 \\ K_2 \\ r_1 \\ r_2 \end{bmatrix}, \qquad \mathrm{H}_0 = \begin{bmatrix} m & 0 \\ 0 & m \end{bmatrix}, \quad \text{and} \quad \mathrm{I}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

8. Solve X from $\mathrm{A}X = B$ to get the accelerations of the object and joints, and the forces exerted on the object.

9. Compute the controlled joint torque τ_i, $i = 1, 2$.

10. Go to Step 1.

BIBLIOGRAPHY

BIBLIOGRAPHY

[ABL88] K. L. Anderson, G. L. Blankenship, and L. G. Lebow. A rule–based adaptive PID controller. In *Proc. of 27th IEEE Conf. on Decision and Control*, pages 564 – 569, 1988.

[APW88] P. J. Antsaklis, K. M. Passino, and S. J. Wang. Autonomous control systems: Architecture and fundamental issues. In *Proc. of 1988 American Control Conf.*, pages 602 – 607, 1988.

[Ara89] M. Arai. Mapping abilities of three–layer neural networks. In *Proc. of Int. Joint Conf. on Neural Networks*, volume 1, pages I419 – I423, 1989.

[Arz89] K.-E. Arzen. Knowledge–based control systems — Aspects on the unification of conventional control systems and knowledge–based systems. In *Proc. of 1989 American Control Conf.*, pages 2233 – 2238, 1989.

[Arz90] K.-E. Arzen. Knowledge–based control systems. In *Proc. of 1990 American Control Conf.*, volume 2, pages 1986 – 1991, 1990.

[AS86] H. Asada and J. J. E. Slotine. *Robot Analysis and Control.* Wiley–Interscience Publication, 1986.

[AS88] T. E. Alberts and D. I. Soloway. Force control of a multi-arm robot system. In *Proc. of 1988 IEEE Conf. on Robotics and Automation*, pages 1490 – 1495, 1988.

[Asa90] H. Asada. Teaching and learning of compliance using neural nets: Representation and generation of nonlinear compliance. In *Proc. of 1990 IEEE Conf. on Robotics and Automation*, volume 2, pages 1237 – 1244, 1990.

[AW90] B. A. Abdulmajid and R. J. Wynne. An improved qualitative controller. In *Proc. of 1990 American Control Conf.*, volume 2, pages 1455 – 1460, 1990.

[Bar89] A. R. Barron. Statistical properties of artificial neural networks. In *Proc. of 28th IEEE Conf. on Decision and Control*, volume 1, pages 280 – 285, 1989.

[Bat89] C. Batur. A real time fuzzy self tuning control. In *Proc. of 1989 American Control Conf.*, pages 1810 – 1815, 1989.

[BB89] D. F. Bassi and G. A. Bekey. High precision position control by cartesian trajectory feedback and connectionist inverse dynamics feedforward. In *Proc. of Int. Joint Conf. on Neural Networks*, volume 2, pages 325 – 331, 1989.

[BCLM89] H. R. Berenji, Y.-Y. Chen, C.-C. Lee, and S. Murugesan. An experiment–based comparative study of fuzzy logic control. In *Proc. of 1989 American Control Conf.*, pages 2751 – 2753, 1989.

[BDG86] D. S. Bernstein, L. D. Davis, and S. W. Greeley. The optimal projection equations for fixed–order sampled–data dynamic compensation with computation delay. *IEEE Trans. on Automatic Control*, 31(9):859 – 862, 1986.

[Bej74] A. K. Bejczy. Robot arm dynamics and control. *JPL Technical Report 33-669*, page 93, Feb. 1974.

[Ber88] J. A. Bernard. Use of a rule–based system for process control. *IEEE Control System Magazine*, pages 3 – 13, Oct. 1988.

[BKG89] I. Bar-Kana and A. Guez. Neuromorphic adaptive control. In *Proc. of 28th IEEE Conf. on Decision and Control*, volume 2, pages 1739 – 1743, 1989.

[BM89] N. Bhat and T. J. McAvoy. Use of neural networks for dynamic modeling and control of chemical process systems. In *Proc. of 1989 American Control Conf.*, volume 2, pages 1342–1347, 1989.

[Cam90] F. A. Camargo. Learning algorithms in neural networks. Technical Report CUCS-062-90 (Draft), The DCC Laboratory, Computer Science Department, Columbia University, Dec. 1990.

[CP89] V. C. Chen and Y. H. Pao. Learning control with neural networks. In *Proc. of 1989 IEEE Conf. on Robotics and Automation*, pages 1448–1453, 1989.

[CS88] X. Cui and K. G. Shin. Robot trajectory tracking with self–tuning predicted control. In *Proc. of 1988 American Control Conf.*, volume 1, pages 529 – 534, 1988.

[Cyb89] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303 – 314, 1989.

[Eld88] H. K. Eldeib. Evaluation of microcomputer–based implementations of knowledge–based control systems. In *Proc. of 27th IEEE Conf. on Decision and Control*, pages 137 – 140, 1988.

[Els88] R. K. Elsley. A learning architecture for control based on back–propagation neural networks. In *Proc. of Int. Conf. on Neural Networks*, volume 2, pages 587 – 594, 1988.

[FAET90] M. S. Fadali, F. J. Aguirre, D. D. Egbert, and E. C. Tacker. Minimum–time control of robotic manipulators using a back propagation neural network. In *Proc. of 1990 American Control Conf.*, volume 3, pages 2997 – 3000, 1990.

[FH88] E. Freund and H. Hoyer. Real–time pathfinding in multirobot systems including obstacle avoidance. *The International Journal of Robotics Research*, 7(1):42 – 70, Feb. 1988.

[Fra89] J. A. Franklin. Historical perspective and state of the art in connectionist learning control. In *Proc. of 28th IEEE Conf. on Decision and Control*, volume 2, pages 1730 – 1736, 1989.

[GEK88] A. Guez, J. L. Eilbert, and M. Kam. Neural network architecture for control. *IEEE Control System Magazine*, pages 22 – 25, April 1988.

[GJ88] Z. Geng and M. Jamshidi. Expert self–learning controller for robot manipulator. In *Proc. of 27th IEEE Conf. on Decision and Control*, pages 1090 – 1095, 1988.

[GP89] M. M. Gupta and W. Pedrycz. Cognitive and fuzzy logic controllers: A retrospective and perspective. In *Proc. of 1989 American Control Conf.*, pages 2245 – 2251, 1989.

[GS88] A. Guez and J. Selinsky. A neuromorphic controller with a human teacher. In *Proc. of Int. Conf. on Neural Networks*, volume 2, pages II595 – II602, 1988.

[Gu90] Y. L. Gu. On nonlinear system invertibility and learning approaches by neural networks. In *Proc. of 1990 American Control Conf.*, volume 3, pages 3013 – 3017, 1990.

[GW88] B. J. Glass and C. M. Wong. A knowledge–based approach to identification and adaptation in dynamical system control. In *Proc. of 27th IEEE Conf. on Decision and Control*, pages 881 – 886, 1988.

[GWG89] M. Gutierrez, J. Wang, and R. Grondin. Estimating hidden unit number for two–layer perceptron. In *Proc. of Int. Joint Conf. on Neural Networks*, volume 1, pages I677 – I681, 1989.

[HA89] E. Hernandez and Y. Arkun. Neural network modeling and an extended dmc algorithm to control nonlinear systems. In *Proc. of 1989 American Control Conf.*, volume 3, pages 2454 – 2459, 1989.

[Hay86] S. Hayati. Hybrid position/force control of multi-arm cooperating robots. In *Proc. of 1986 IEEE Conf. on Robotics and Automation*, pages 82 - 89, 1986.

[HF91] E. T. Hancock and F. Fallside. A direct control method for a class of nonlinear systems using neural networks. Technical Report CUED/F-INFENG/TR.65, Cambridge University, Engineering Department, Cambridge, England, March 1991.

[HN89] R. H.-Nielsen. Theory of the backpropagation neural networks. In *Proc. of Int. Joint Conf. on Neural Networks*, volume 1, pages I593 - I605, 1989.

[Hsu89] P. Hsu. Control of multi-manipulators systems — Trajectory tracking, load distribution, internal force control, and decentralized architecture. In *Proc. of 1989 IEEE Conf. on Robotics and Automation*, pages 1234 - 1239, 1989.

[IS90] S. Isaka and A. V. Sebald. An optimization approach for fuzzy controller design. In *Proc. of 1990 American Control Conf.*, volume 2, pages 1485 - 1490, 1990.

[Isi87] C. Isik. Identification and fuzzy rule-based control of a mobile robot motion. In *Proc. of IEEE The 2nd Inte. Symp. on Intelligent Control*, pages 94 - 99, 1987.

[JD87] J. Jiang and R. Doraiswami. Information acquisition in expert control system design using adaptive filters. In *Proc. of IEEE The 2nd Inte. Symp. on Intelligent Control*, pages 165 - 170, 1987.

[JL90] M. A. Johnson and M. Leahy. Adaptive model-based neural network control. In *Proc. of 1990 IEEE Conf. on Robotics and Automation*, volume 3, pages 1704 - 1709, 1990.

[JS88] J. R. James and G. J. Suski. A survey of some implementations of knowledge-based systems for real-time control. In *Proc. of 27th IEEE Conf. on Decision and Control*, pages 580 - 585, 1988.

[KB88] A. J. Koivo and G. A Bekey. Report of workshop on coordinated multiple robot manipulators: Planning, control and applications. *IEEE J. of Robotics and Automation*, 4(1):91 - 93, Feb. 1988.

[KBVB88] A. J. Krijgsman, H. M. T. Broeders, H. B. Verbruggen, and P. M. Bruijn. Knowledge-based control. In *Proc. of 27th IEEE Conf. on Decision and Control*, pages 570 - 574, 1988.

[KC81] R.M.C.D. Keyser and A.R.V. Cauwenberghe. A self-tuning multistep predictor application. *Automatica*, 17(1):167 - 174, 1981.

[KC89] L. G. Kraft and D. P. Campagna. A comparison of CMAC neural network and traditional adaptive control. In *Proc. of 1989 American Control Conf.*, volume 1, pages 884 – 889, 1989.

[KH88] S. Y. Kung and J. N. Hwang. An algebraic projection analysis for optimal hidden units size and learning rates in BP learning. In *Proc. of Int. Conf. on Neural Networks*, volume 1, pages I363 – I370, 1988.

[KH89] S. Y. Kung and J. N. Hwang. Neural network architectures for robotic applications. *IEEE Tran. on Robotics and Automation*, 5(5):641 – 657, Oct. 1989.

[Kha86] O. Khatib. Real time obstacle avoidance for manipulator and mobile robots. *International Journal of Robotics Research*, 5(1):90 – 98, 1986.

[Kho87] P. K. Khosla. Choosing sampling rates for robot control. In *Proc. of 1987 IEEE Conf. on Robotics and Automation*, pages 169 – 174, 1987.

[KKG88] M. M. Kokar, S. N. Keshav, and S. Gopalraman. Learning in semantic control. In *Proc. of 27th IEEE Conf. on Decision and Control*, pages 1812 – 1817, 1988.

[KL88] K. Kreutz and A. Lokshin. Load balancing and closed chain multiple arm control. In *Proc. of 1988 American Control Conf.*, pages 2148 – 2153, 1988.

[Koh87] T. Kohonen. State of the art in neural computing. In *Proc. of Int. Conf. on Neural Networks*, volume 1, pages I79 – I90, 1987.

[Koi85] A. J. Koivo. Adaptive position–velocity–force control of two manipulators. In *Proc. of 24th IEEE Conf. on Decision and Control*, pages 1529 – 1532, 1985.

[KT88] H. Kazerooni and T. I. Tsay. Compliance control and unstructured modeling of cooperating robots. In *Proc. of 1988 IEEE Conf. on Robotics and Automation*, pages 510 – 515, 1988.

[KV89] H. Kang and G. Vachtsevanos. Model reference fuzzy control. In *Proc. of 28th IEEE Conf. on Decision and Control*, volume 1, pages 751 – 756, 1989.

[KY88] C. D. Kopf and T. Yabuta. Experimental comparison of master/slave and hybrid two arm position/force control. In *Proc. of 1988 IEEE Conf. on Robotics and Automation*, pages 1633 – 1638, 1988.

[Lee90] C. C. Lee. Fuzzy logic in control systems: Fuzzy logic controller — Part I and II. *IEEE Trans. on Systems, Man, and Cybernetics*, pages 404 – 435, March 1990.

[LGI89] E. Levin, R. Gewirtzman, and G. F. Inbar. Neural network architecture for adaptive system modeling and control. In *Proc. of Int. Joint Conf. on Neural Networks*, volume 2, pages 311 – 316, 1989.

[Lip87] R. P. Lippmann. An introduction to computing with neural networks. *IEEE ASSP Magazine*, pages 4 – 22, April 1987.

[Lit90] J. Litt. An expert system to perform on–line controller tuning. In *Proc. of 1990 American Control Conf.*, volume 2, pages 1968 – 1973, 1990.

[LK87] S. Lee and M. H. Kim. Cognitive control of dynamic systems. In *Proc. of IEEE The 2nd Inte. Symp. on Intelligent Control*, pages 455 – 460, 1987.

[LL84] C. S. G. Lee and B. H. Lee. Resolved motion adaptive control for mechanical manipulators. *ASME Journal of Dynamics, Measurement and Control*, 106(2):134 – 142, June 1984.

[LL89] Y. F. Li and C. C. Lau. Development of fuzzy algorithms for servo systems. *IEEE Control System Magazine*, pages 65 – 72, April 1989.

[LMO82] R. E. Larson, P. L. McEntire, and J. G. O'Reilly, editors. *Tutorial: Distributed Control.* IEEE Computer Society, 2 edition, 1982.

[LZ88] J. Y. S. Luh and Y. F. Zheng. Load distribution between two coordinating robots by nonlinear programming. In *Proc. of 1988 American Control Conf.*, pages 479 – 482, 1988.

[MB89] L. Mo and M. M. Bayoumi. Adaptive control of the multi-arm robotic system. In *Proc. of 28th IEEE Conf. on Decision and Control*, pages 1962 – 1963, 1989.

[Mel89] P. J. W. Melsa. Neural networks: A conceptual overview. Technical Report TRC-89-08, Tellabs Research Center, Aug. 1989.

[Mey87] A. Meystel. Intelligent control: Highlights and shadows. In *Proc. of IEEE The 2nd Inte. Symp. on Intelligent Control*, pages 1 – 8, 1987.

[MHGK90] W. T. Miller, R. P. Hewes, F. H. Glanz, and L. G. Kraft. Real–time dynamic control of an industrial manipulator using a neural–network–based learning controller. *IEEE Trans. on Robotics and Automation*, 6(1):1 – 9, Feb. 1990.

[Mit85] T. Mita. Optimal digital feedback control systems counting computation time of control laws. *IEEE Trans. on Automatic Control*, 30(6):542 – 547, June 1985.

[MW90] M. Marcos and P. E. Wellstead. An intelligent rule–based compensator for control system input nonlinearities. In *Proc. of 1990 American Control Conf.*, volume 2, pages 1467 – 1473, 1990.

[Nak88] Y. Nakamura. Minimizing object strain energy for coordination of multiple robotic mechanisms. In *Proc. of 1988 American Control Conf.*, pages 499 – 504, 1988.

[NKR89] G. W. Neast, H. Kaufman, and R. J. Roy. Expert adaptive control for drug delivery systems. *IEEE Control System Magazine*, pages 20 – 23, June 1989.

[NP90] K. S. Narendra and K. Parthasarathy. Identification and control of dynamic systems using neural networks. *IEEE Trans. on Neural Networks*, 1(1):4 – 27, March 1990.

[OS88] R. Ortega and M. W. Spong. Adaptive motion control of rigid robots: A tutorial. In *Proc. of 27th IEEE Conf. on Decision and Control*, pages 1575 – 1584, 1988.

[Ozg89] U. Ozguner. Decentralized and distributed control approached and algorithms. In *Proc. of 28th IEEE Conf. on Decision and Control*, volume 2, pages 1289 – 1294, 1989.

[Pit88] M. E. Pittelkau. Adaptive load–sharing force control for two–arm manipulators. In *Proc. of 1988 IEEE Conf. on Robotics and Automation*, pages 498 – 503, 1988.

[PJM87] B. Porter, A. H. Jones, and C. B. McKeown. Real–time expert controller for plants with actuator non-linearities. In *Proc. of IEEE The 2nd Inte. Symp. on Intelligent Control*, pages 171 – 177, 1987.

[Pou89] F. Pourboughrat. Neuromorphic controllers. In *Proc. of 28th IEEE Conf. on Decision and Control*, volume 2, pages 1748 – 1749, 1989.

[PSY88] D. Psaltis, A. Sideris, and A. A. Yamamura. A multilayered neural network controller. *IEEE Control System Magazine*, pages 17 – 21, April 1988.

[RM86] D. E. Rumelhart and J. L. McCelland. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, volume 1: Foundations.* MIT Press, 1986.

[Sar88] G. N. Saridis. On the theory of intelligent machines: A survey. In *Proc. of 27th IEEE Conf. on Decision and Control*, pages 1799 – 1804, 1988.

[SC88] K. G. Shin and X. Cui. Effects of computing time delay on real–time control systems. In *Proc. of 1988 American Control Conf.*, volume 2, pages 1071 – 1076, 1988.

[Ser87] H. Seraji. A approach to multivariable control of manipulators. *ASME Journal of Dynamics, Measurement and Control*, 109:146 - 154, Sept. 1987.

[Ser88] H. Seraji. Coordinated adaptive position/force control of dual–arm robots. In *a manuscript submitted for publication*, 1988.

[SKL85] K. G. Shin, C. M. Krishna, and Y.-H. Lee. A unified method for evaluating real–time computer controller and its application. *IEEE Trans. on Automatic Control*, 30(4):357 - 366, April 1985.

[Son90] E. D. Sontag. Feedback stabilization using two–hidden–layer nets. *Technique report of Rutgers Center for Systems and Control, No. SYCON-90-1*, Oct. 1990.

[SW89] M. Stinchcombe and H. White. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In *Proc. of Int. Joint Conf. on Neural Networks*, volume 1, pages I613 - I617, 1989.

[Swi89] R. W. Swiniarski. Novel neural network based self–tuning PID controller which uses pattern recognition. In *Proc. of 1989 American Control Conf.*, volume 3, pages 3023 - 3024, 1989.

[SZ89] K. G. Shin and Q. Zheng. Minimum time trajectory planning for dual robot systems. In *Proc. of 28th IEEE Conf. on Decision and Control*, pages 2506 - 2511, 1989.

[TBY86] T. J. Tarn, A. K. Bejczy, and X. Yun. Coordinated control of two robot arms. In *Proc. of 1986 IEEE Conf. on Robotics and Automation*, pages 1193 - 1202, 1986.

[TBY87] T. J. Tarn, A. K. Bejczy, and X. Yun. Design of dynamic control of two cooperating robot arms: Closed chain formulation. In *Proc. of 1987 IEEE Conf. on Robotics and Automation*, pages 7 - 13, 1987.

[TBY88] T. J. Tarn, A. K. Bejczy, and X. Yun. New nonlinear control algorithms for multiple robot arms. *IEEE Tran. on Aerospace and Electronic Systems*, 24(5):571 - 582, Sept. 1988.

[Til90] R. B. Tilove. Local obstacle avoidance for mobile robots based on the method of artificial potential. In *Proc. of 1990 IEEE Conf. on Robotics and Automation*, pages 566 - 571, 1990.

[TY86] M. Togai and O. Yamano. Learning control and its optimality: Analysis and its application to controlling industrial robots. In *Proc. of 1986 IEEE Conf. on Robotics and Automation*, volume 1, pages 248 - 253, 1986.

[Uch87] M. Uchiyama. Hybrid position/force control for coordination of a two arm robot. In *Proc. of 1987 IEEE Conf. on Robotics and Automation*, pages 1242 - 1247, 1987.

[UD88] M. Uchiyama and P. Dauchez. A symmetric hybrid position/force control scheme for the coordination of two robots. In *Proc. of 1988 IEEE Conf. on Robotics and Automation*, pages 350 - 355, 1988.

[Wal86] M. W. Walker. The course notes of eecs567. the Department of Electrical Engineering and Computer Science, the University of Michigan, 1986.

[War90] C. W. Warren. Multiple robot path coordination using artificial potential fields. In *Proc. of 1990 IEEE Conf. on Robotics and Automation*, volume 1, pages 500 - 505, 1990.

[Wer88] P. J. Werbos. Backpropagation: Past and future. In *Proc. of Int. Conf. on Neural Networks*, volume 1, pages I343 - I353, 1988.

[Wer89a] P. J. Werbos. Backpropagation and neurocontrol: A review and prospectus. In *Proc. of Int. Joint Conf. on Neural Networks*, volume 1, pages I209 - I216, 1989.

[Wer89b] P. J. Werbos. Neural networks for control and system identification. In *Proc. of 28th IEEE Conf. on Decision and Control*, volume 1, pages 260 - 265, 1989.

[Whi89] J. E. Whitlow. A knowledge-based structure for process control. In *Proc. of 1989 American Control Conf.*, pages 1354 - 1359, 1989.

[WHR90] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart. Predicting the future: A connectionist approach. *Int. Journal of Neural Systems*, 1(3):193 - 209, 1990.

[WKD89] M. W. Walker, D. Kim, and J. Dionise. Adaptive coordinated motion control of two manipulator arms. In *Proc. of 1989 IEEE Conf. on Robotics and Automation*, pages 1084 - 1090, 1989.

[WM89] J. Wang and B. Malakooti. On training of artificial neural networks. In *Proc. of Int. Joint Conf. on Neural Networks*, volume 2, pages 387 - 393, 1989.

[Woo86] M. H. Woodbury. Analysis of execution time of real-time tasks. In *Proc. of 1986 IEEE Real-Time Systems Symp.*, pages 89 - 96, 1986.

[YG89] D.-Y. Yeung and G. A. Gekey. Using a context-sensative learning network for robot arm control. In *Proc. of 1989 IEEE Conf. on Robotics and Automation*, pages 1441 - 1447, 1989.

[Yun89] X. Yun. Nonlinear feedback control of two manipulators in presence of environmental constraints. In *Proc. of 1989 IEEE Conf. on Robotics and Automation*, pages 1252 – 1257, 1989.

[YY90] T. Yabuta and T. Yamada. Possibility of neural networks controller for robot manipulators. In *Proc. of 1990 IEEE Conf. on Robotics and Automation*, pages 1686 – 1691, 1990.

[ZL85] Y. F. Zheng and J. Y. S. Luh. Control of two coordinated robots in motion. In *Proc. of 24th IEEE Conf. on Decision and Control*, pages 1761 – 1766, 1985.

[ZL88] Y. F. Zheng and J. Y. S. Luh. Optimal load distribution for two industrial robot handling a single object. In *Proc. of 1988 IEEE Conf. on Robotics and Automation*, pages 344 – 349, 1988.

[ZPK89] V. Zeman, R. V. Patel, and K. Khorasani. A neural network based control strategy for flexible–joint manipulators. In *Proc. of 28th IEEE Conf. on Decision and Control*, volume 2, pages 1759 – 1764, 1989.