The University of Zambia
School of Engineering
Department of Electrical & Electronic Engineering

Intelligent Control of the Inverted Pendulum

by

Anthony Ng’oma

A thesis submitted to the University of Zambia in fulfilment of
e requirements for the Degree of Master of Engineering ”

July, 1997

DECLARATION

I, Anthony Ng'oma, do solemnly declare that this dissertation represents my own work and that it has not been submitted for a degree at the University of Zambia or any other University.

Signature : [illegible]

Date : 24/02/98

APPROVAL

This dissertation of Anthony Ng'oma is approved as fulfilling the requirements for the award of the Degree of Master of Engineering (Electrical) by the University of Zambia.

EXAMINER'S SIGNATURE	DATE
[illegible]	
[illegible]	23/3/98
[illegible]	17/4/1998

ABSTRACT

Artificial Intelligence (AI) based control techniques are becoming more popular for their applicability to complex control systems. In this research, an Intelligent Controller is designed for a non-linear process with complex dynamics - the Inverted Pendulum. The controller is designed to interact with other controllers in the process. That is, the Intelligent Controller is designed to control the Inverted Pendulum for part of the state space and afterwards transfers control to other existing controllers. This arrangement is referred to as Operation-Space (OS) based control. The Intelligent Controller is designed to steer the Inverted Pendulum from the downward resting position or state, to a state controllable by an existing OS based controller. Control knowledge for this swing up operation is generated by applying a multi-stage decision process called Differential Dynamic Programming (DDP).

A DDP algorithm is developed by solving a set of partial differential equations. The Inverted Pendulum is represented as a discretised fourth order process model. The discretised model is based on the corrected van Luenen continuous time model. The van Luenen model has an error which I discovered using step response simulations. After a series of trials, two cost objective functions J_1 and J_2 were chosen for the DDP algorithm. This was the most difficult part of the design process. The DDP equations were implemented in a computer algorithm to compute optimal control data in the form of process states and control signals. The downward resting position was the initial state , $x(0)$ while the final state $x(N)$, was a state controllable by the OS based controller. The resulting two sets of control data constituted the training data for the Neural Network (N-Net) structure.

An N-Net structure was designed and trained on the generated control data. Its simulation results showed excellent performance. Sensitivity analysis to variations of model parameters was performed. The N-Net was then implemented as a controller on the laboratory setup. Hardware constraints on the laboratory setup and the transputer system, made it difficult to test the controller's full capabilities. However, results obtained indicated that the N-Net controller was able to reproduce the control trajectories in real time.

In addition to the design of the N-Net controller, an integrated PD swing up controller was designed and implemented on the laboratory setup. The PD controllers, assisted by saturation effects, successfully steer the Inverted Pendulum from the downward resting position to join the OS based control cycle. The controller demonstrates compatibility with the existing OS based controller, which the N-Net was designed to have.

PREFACE

This M.Eng Thesis is a product of research collaboration between the Department of Electrical & Electronic Engineering of the University of Zambia (UNZA), and the Control Laboratory of the Technical University of Twente (UT), in The Netherlands. The hardware and software equipment were acquired from the control laboratory. Research was initiated in the Netherlands and completed in Zambia. This Thesis was prepared while at the University of Zambia.

The AI technology is relatively a young discipline. I had little knowledge in the field at the start of my research work. During my stay in the Netherlands, I followed some advanced courses offered by the Dutch Institute of Systems and Control (DISC), which introduced me to the subject. My interaction with MSc and PhD students at the UT and the Technical University of Delft (TU Delft) also helped me a lot. In addition, I was privileged to attend some international conferences in Systems and Control. This provided me with more understanding of the AI technology and its applicability in industry. After working with AI techniques in this research and otherwise, my opinion of the technology is that even though it does not promise to solve ALL our problems, the methods have a great potential in solving many complex control problems, as has been demonstrated by the results of this research. If applied appropriately, the techniques could change so many things for the better, in the Zambian Industries.

A. Ng'oma
July, 1997

ACKNOWLEDGEMENTS

I would like to thank my supervisors Prof. dr. ir. Job van Amerongen and dr. ir. Theo de Vries both of the University of Twente (UT), The Netherlands, and Mr. D. Beekman of the University of Zambia (UNZA), for their guidance in my work. Thanks also to Erik Coelingh (UT) for his valuable advice and to Roger Bruis (UT) for the technical support.

I also wish to express my gratitude to Prof. Andre Bakkers (UT) who introduced me to Parallel Processing and the Occam Programming Language. In addition, I thank Dr. L. D. Nyirenda (UNZA) for the literature material and moral support rendered to me, Dr. A. J. Mouthaan (UT/UNZA) and Mr. Lo-A-Njoe (UNZA) for their efforts in facilitating the smooth running of my work.

Further, I wish to acknowledge the assistance received under the MHO project. Finally, I thank my dear wife, Ivy, for her patience and all the support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

SYMBOLS

x	State vector
φ , θ	Angular displacement in [rad or deg]
$\dot{\varphi}$, $\dot{\theta}$	Angular speed in [rad/s or deg/s]
R_c, R_s, R_d	Friction terms in [Nms]
J_s, J_d	Moment of inertia in [Kgm^2]
m_s, m_d	Mass [Kg]
h	Sample interval [s]
l_s, l_b	Length [m]
γ , α	Proportionality constant - learning rate
J	Cost function (objective function)
u(k)	Control signal

ABBREVIATIONS

N-Net	Neural Network
MSE	Mean Squared Error
OS	Operation-Space
ANN	Artificial Neural Network
AI	Artificial Intelligence
LQ	Linear Quadratic
NL	Non-linear

DDP	Differential Dynamic Programming
FF	Feed-Forward

1.0 INTRODUCTION

1.1 Background

In recent years, there has been renewed interest in "Artificial Intelligence" (AI) based methods for solving control system problems and other tasks. AI based techniques are attractive because of their ability to solve complex control problems. Non-linear dynamic systems present challenging control problems. The Inverted Pendulum (or balancing arm) is an example of a fourth order non-linear dynamic system. It is often used in investigations of new control techniques. The Inverted Pendulum belongs to the class of under-actuated mechanical systems. Under-actuated systems are those systems which possess fewer actuators than degrees of freedom. Complex internal dynamics and lack of feedback linearization are often exhibited by such systems, making the class a rich one from the control standpoint [1]. In this research the application of AI methods to the swing up control problem of the Inverted Pendulum is studied. The experimental planar setup has two-degrees of freedom with only one actuator. The UNZA setup is a replica of the Operation-Space (OS) based controlled setup at the UT control laboratory [2].

At the control laboratory of the University of Twente (UT) research is carried out towards the application of AI methods for control of dynamic systems. An Inverted Pendulum setup has been developed and is used as a non-linear process for research work. An Operation-Space (OS) based controller has been developed and implemented on the laboratory setup. The controller is a product of previous research work done by Oosterveen [3], Jongma [4] and van Luenen [5]. Oosterveen developed the Inverted Pendulum model equations and identified the model parameters on the real setup. He also developed the pendulum's linearised model and designed a state feedback controller for the linear state space. Jongma and van Luenen designed and implemented a non-linear B-Spline network controller to restore control of the pendulum after failure of the LQ controller. The controllers were combined in an OS based controller setup (see chapter 3). The OS based controller operates successfully only in limited state space (see Section 4.1.2). In this research a new OS-based controller is developed by designing and implementing a fourth controller in the OS-based controller. The controller is a non-linear controller implemented in form of a N-Net. The new OS based controller has the ability to swing the pendulum from rest (stationary position) before controlling it in the vertical position. In addition, various N-Net training methods are studied and the results used in the controller design.

In Zambia, the need for improved control techniques is great. There are many industrial plants running complex processes. Examples include the Nitrogen Chemicals of Zambia (NCZ), Indeni Oil Refinery, Zambia Consolidated Copper Mines (ZCCM), Kabwe Industrial Fabrics Company (KIFCO), Mulungushi Textiles and many others. AI based techniques could in future be utilised to realise more effective performance in their chemical process and other control systems.

1.2 Thesis goals

The main goal of this research is to design an AI based non-linear swing up controller for the Inverted Pendulum. The problem is tackled through a number of sub goals which include:

- To determine the best controller design approach to allow for compatibility of the new controller with the existing OS based controller or parts of it.
- To develop and implement a Differential Dynamic Programming algorithm to generate the desired control knowledge.

- To design a Neural Network structure and investigate the suitability of different learning algorithms to the same.
- To assess the performance of the controller in simulations.
- To implement the controller on the Inverted Pendulum laboratory setup.

1.3 Artificial Intelligence (AI) for Control

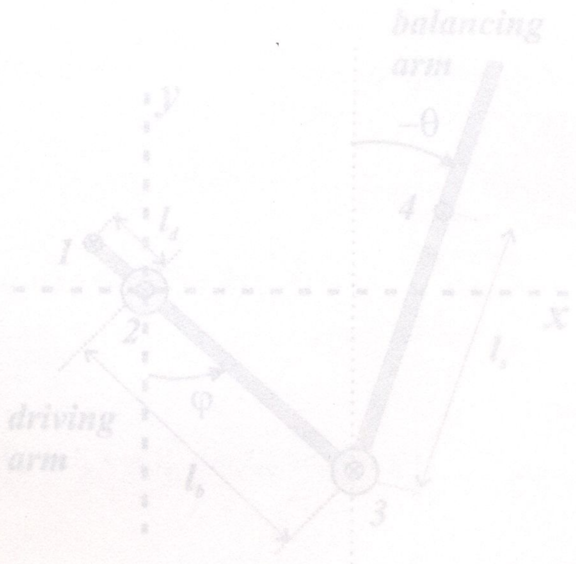
Currently, research in AI methods is in two main directions. The first approach is based on the idea that human intelligence can be reproduced by imitating man's reasoning capabilities. This is referred to as Symbolic AI and it encompasses fuzzy logic. The second approach, subsymbolic AI, is based on the concept of learning, seen as a function of the brain. This concept has led to the development of Artificial Neural Network (ANN) structures [6].

AI methods provide many possibilities which are of special interest in the development of control systems or incorporation in the same. The technique of Artificial Neural Networks, born from the subsymbolic AI concept, offers tools to implement complex non-linear mappings. The technique can be used to implement non-linear (dynamic) input-output behaviours such as in the implementation of NARMAX (Non-linear AutoRegressive Moving Average eXogenous) models. This method offers a great advantage in achieving high precision due to the fact that continuous mappings are possible [6]. The Intelligent Controller was designed using this technique. Other possible methods for the swing up controller may include open loop strategies, sinusoidal excitation and control of system energy. Partial feedback linearization and integrator back stepping, used by Spong [1] in the swing up control problem for the Acrobot, may also be used.

AI methods are also applied in the incorporation of uncertainty in systems and also in plant-wide control schemes [6][7][8].

1.4 The Inverted Pendulum

The Inverted Pendulum under study in this research comprises two arms [4]: the driving arm and the balancing arm as shown in Figure 1.1. The driving arm of is shorter and is connected to a shaft on the driving arm axis. The balancing arm connects to the other end of the driving arm by the balancing arm axis. The shaft is driven by a DC motor. The balancing arm axis is parallel to the driving arm axis, such that both arms are able to rotate only in the plane perpendicular to both axes. The lengths of the driving and balancing arms are given by l_d and l_s, respectively.



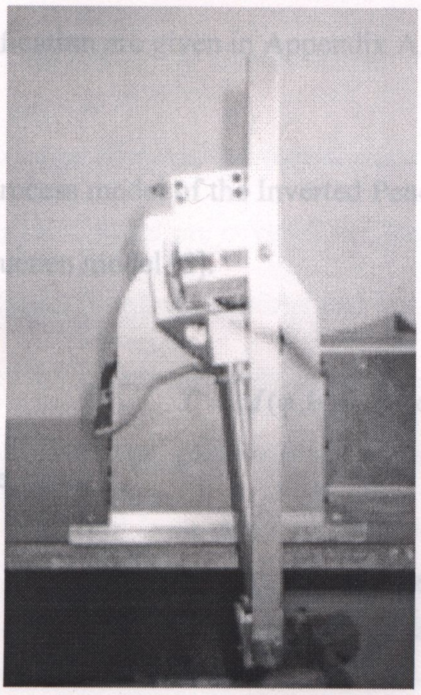
1. centre of mass of driving arm
2. driving arm axis
3. balancing arm axis
4. centre of mass of balancing arm

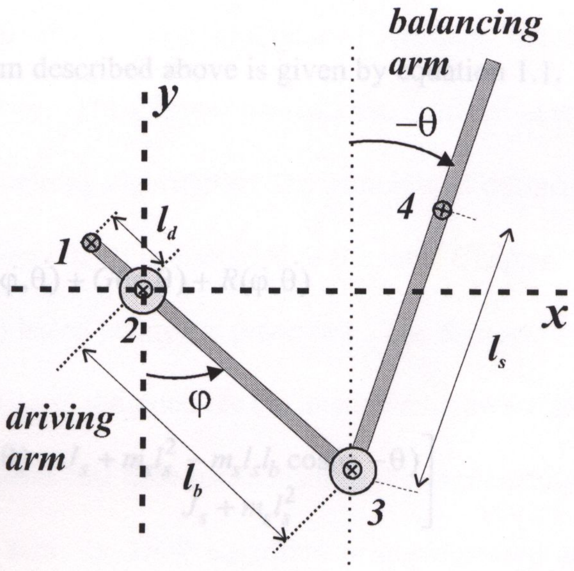
(a) (b)

Figure 1.1 The Inverted Pendulum: (a) picture and (b) definition of angles

1.4 *The Inverted Pendulum*

The Inverted Pendulum under study in this research comprises two arms [4]: the driving arm and the balancing arm as shown in Figure 1.1. The driving arm of is shorter and is connected to a shaft on the driving arm axis. The balancing arm connects to the other end of the driving arm by the balancing arm axis. The shaft is driven by a DC motor. The balancing arm axis is parallel to the driving arm axis, such that both arms are able to rotate only in the plane perpendicular to both axes. The lengths of the driving and balancing arms are given by l_d and l_s, respectively.





1. centre of mass of driving arm
2. driving arm axis
3. balancing arm axis
4. centre of mass of balancing arm

(a) (b)

Figure 1.1 The Inverted Pendulum: (a) picture and (b) definition of angles

The Inverted Pendulum is controlled by applying a torque to the motor axis. There are two resolvers, one on the balancing arm axis and the other on the driving arm axis. The resolvers measure the angles of the two arms, θ and φ, respectively. The masses of the driving arm and the balancing arm are represented by m_d and m_s, respectively. J_d is the inertial moment of the driving arm on its axis. J_s is the inertial moment of the balancing arm on its axis. Three friction terms are defined: the viscous friction on the driving arm axis R_d; the viscous friction on the balancing arm axis R_s; and non-linear friction (coulomb friction or stiction) on the driving axis R_c. The control signal is the torque T1 on the driving axis. The estimates of these model parameters after identification are given in Appendix A.

The process model of the Inverted Pendulum described above is given by equation 1.1. This is the van Luenen model [5].

$$T = M(\varphi,\theta)\ddot{v} + C(\varphi,\theta,\dot{\varphi},\dot{\theta}) + G(\varphi,\theta) + R(\dot{\varphi},\dot{\theta}) \tag{1.1}$$

Where

$$M = \begin{bmatrix} J_d + m_s l_b^2 - m_s l_s l_b \cos(\varphi - \theta) & J_s + m_s l_s^2 - m_s l_s l_b \cos(\varphi - \theta) \\ - m_s l_s l_b \cos(\varphi - \theta) & J_s + m_s l_s^2 \end{bmatrix}$$

$$C = \begin{bmatrix} - m_s l_s l_b \sin(\varphi - \theta)\dot{\theta}^2 + m_s l_s l_b \sin(\varphi - \theta)\dot{\varphi}^2 \\ m_s l_s l_b \sin(\varphi - \theta)\dot{\varphi}^2 \end{bmatrix}$$

$$G = \begin{bmatrix} - g(l_d m_d + l_b m_s)\cos\varphi - g l_s m_s \cos\theta \\ - g l_s m_s \cos\theta \end{bmatrix}$$

$$R = \begin{bmatrix} (R_d + R_s)\dot{\theta} - R_c sign\dot{\varphi} \\ - R_s\dot{\varphi} + R_s\dot{\theta} \end{bmatrix}$$

$$\ddot{v} = \begin{bmatrix}\ddot{\varphi} & \ddot{\theta}\end{bmatrix}^T$$

and

$$T = \begin{bmatrix}T1 & T2\end{bmatrix}^T$$

The control signal T1 is limited between -1.97 Nm and 1.97 Nm. T2 = 0 because there is no actuation on the balancing arm.

1.5 Outline of the thesis

Chapter one presents the introduction to the thesis. The principles of Artificial Intelligence and Intelligent Control are covered in Chapter two. The Chapter presents the artificial neural network (ANN) and B-spline networks and various training algorithms. The principle of optimal control as applied in Differential Dynamic Programming (DDP) is discussed in the same Chapter. In Chapter three, the principle of operation-space (OS) based control is presented. The different components of the OS based controller are described. Also highlighted are the available hardware and software resources; and how they are utilised to achieve real time parallelism in the implementation of the existing OS based controller. In Chapter four, the DDP algorithm is implemented and used to compute optimal control data. The design of the combined PD swing up controller is also presented. The combined PD controller's compatibility with the existing OS based controller is assessed. The design of an N-Net structure is presented. The N-Net structure is trained on the optimal control data generated. This controller and the PD swing up controller are implemented on the laboratory setup in Chapter five. In Chapter six, simulation and experimental results of the combined PD and the N-Net controllers are presented. The report ends with conclusions and recommendations in Chapter seven.

2.0 INTELLIGENT CONTROL

An intelligent control system is a system that has the ability to comprehend, reason, and learn about processes, disturbances and operating conditions. Factors that have to be understood and learned are primary process characteristics like static and dynamic behaviour among other things [9]. The desire is to be able to acquire this knowledge and store it in such a way that it can be used and easily retrieved. In general, knowledge may be stored in form of rules, using fuzzy logic principles, or in form of knowledge structures such as in an N-Net. In this chapter, the basic principles behind N-Nets and the combination with fuzzy logic are presented.

2.1 The Artificial Neural Network

The Artificial Neural Network(ANN) concept is derived from the understanding that the human brain comprises numerous tiny units called neurons which are heavily interconnected. The resulting network processes information in parallel. The ANN therefore, consists of a pool of simple processing units (neurons) which communicate by sending signals to each other over a large number of weighted connections. Each processing unit receives an input either from neighbouring units or external sources, computes the output signal and later propagates it to other units [7][10][11][12]. Each unit also adjusts the weights. Figure 2.1 below shows the components that make up the simple artificial neuron, while Figure 2.2 shows the basic structure of an ANN. An N-Net system may consist of three types of units, namely: input; output; and hidden units. Input units receive data from outside, while the output units send it outside the network. For hidden units, their inputs and outputs remain within the system itself.

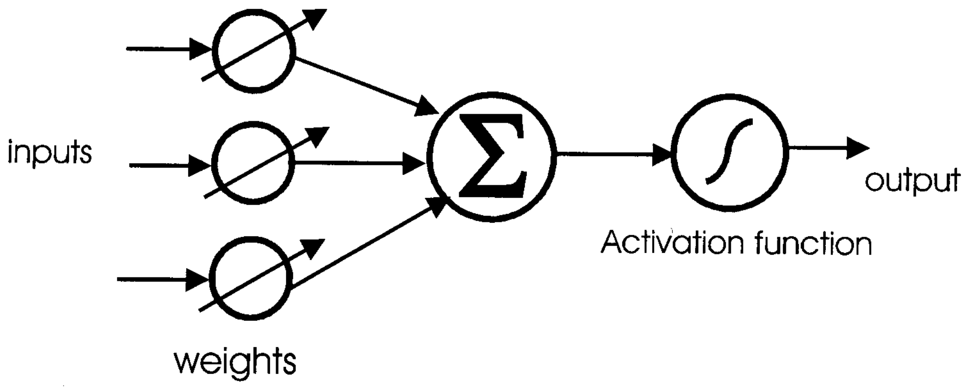


Figure 2.1 An artificial neuron

During operation, units can be updated either **synchronously** or **asynchronously**. With synchronous updating, all units update their activation simultaneously. As for asynchronous updating, each unit has a (usually fixed) probability of updating its activation at a time t, and usually only one unit will be able to do this at a time. The former model is more widely applied although in some cases the latter model has some advantages [10].

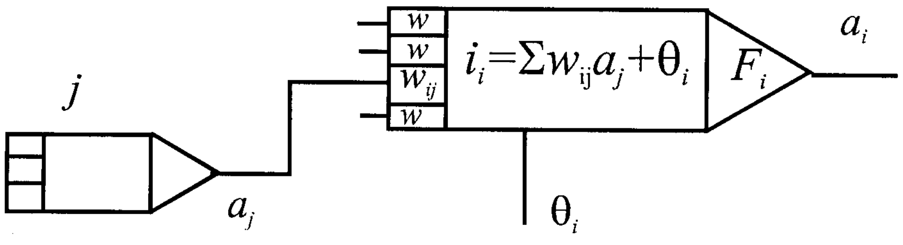


Figure 2.2 Basic components of an ANN

The symbols in Figure 2.2 are:

- a_i State of activation for every unit. It also determines the output of the unit.
- w_{ij} Weight defining each connection. It also determines the effect which the signal of unit ***j*** has on unit ***i***.
- θ_i External input or offset for each unit.

- F_i Activation function, which determines the new level of activation based on the current state ***a**$_i$**(t)**.*
- i_i Effective input to the *i*th neuron. This is a summation of all inputs according to the following relations:

$$i_i = \sum_j w_{ij} a_j + \theta_i \qquad \textbf{Standard} \text{ weighted propagation rule} \qquad (2.1)$$

$$i_i = \sum_j w_{ij} a_j \prod_k a_{jn} + \theta_i \qquad \textbf{Sigma-pi} \text{ unit propagation rule} \qquad (2.2)$$

The activation function F_i takes the total input and the current activation a_i *(t)* to compute a new value of the activation unit ***i***:

$$a_i(t+1) = F_i\big(a_i(t), i_i(t)\big) \qquad (2.3)$$

Often the activation function is a non decreasing function of the total input of the unit:

$$a_i(t+1) = F_i\big(i_i(t)\big) = F_i\left(\sum w_{ij} a_j + \theta_i\right) \qquad (2.4)$$

Three examples of threshold activation functions are: the hard limiting function (e.g. **sgn** function); the linear/semi-linear function; and the smoothly limiting threshold function, such as the **sigmoid** (S-shaped) function of the form:

$$a_i = F_i(i_i) = \frac{1}{1+e^{-i_i}} \qquad (2.5)$$

Figure 2.3 below shows the three examples of threshold activation functions mentioned. There are a few variations in the pattern of connections between the units and the propagation of the data. The main distinctions in connection patterns are **feed-forward (FF)** and **recurrent** networks. In feed-forward topologies, data flow is strictly "one way"- from the input to the output units and there are no feedback connections present. Examples of FF topologies include the **Perceptron** and the **Aldaline** [12]. However, in the case of recurrent networks, feedback connections do exist.

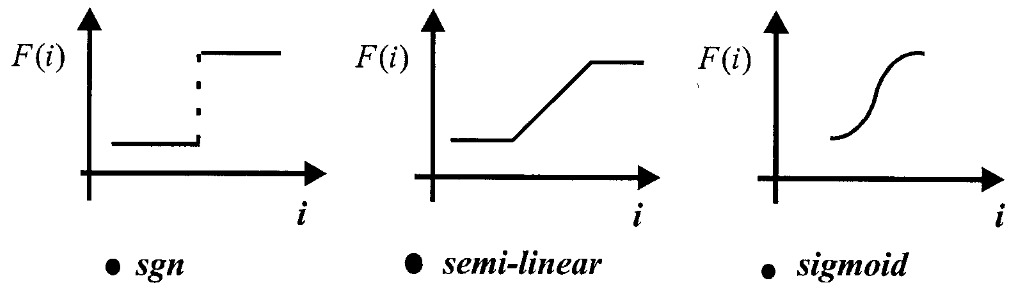


Figure 2.3 Three examples of threshold activation functions

An N-Net is configured in such a way that when a set of inputs is applied, the corresponding desired outputs are achieved. Several methods to set the strengths of the connections exist. One method is to set the weights explicitly, using a priori knowledge. Another way is to 'train' the N-Net by applying training data patterns and allowing the network to adjust its weights in order to meet the target outputs. Adjustment of weights is done according to a variation of "learning" rules. The process of weight adjustment to effectively reduce the error made on the network output value is what is referred to as **learning**. Usually learning is categorised into two distinct types, namely: **supervised** (associative) learning, and **unsupervised** (self organisation) learning. In supervised learning the network is trained by applying input data and its matching output patterns (targets). This is also known as learning from examples. Unsupervised learning involves training an output unit to respond to clusters of pattern within the input. The system must develop its own representation of the input stimuli [10].

The learning process is carried out in accordance with some learning rules. Some common learning rules include: the delta rule (Widrow-Hoff rule), the Perceptron learning rule, back propagation learning rule, Levenberg-Marquardt rule, and others [10][12][13].

2.2 *Multi-layered Neural Networks*

It has already been stated that an N-Net may consist of input, output and hidden units to form several layers. There are no connections within a layer. Single layer networks are not able to carry out any transformation. A multi-layer N-Net shown in Figure 2.4 is capable of solving a much wider range of problems. A multi-layered network with only one layer of hidden units is sufficient to approximate any function with finitely many discontinuities to arbitrary precision, provided that the activation functions of the hidden units are non-linear [10][14]. Multi-layered networks with hidden layers are therefore, more useful, and are usually used in many AI applications. Figure 2.4 shows an N-Net with *l* layers of units.

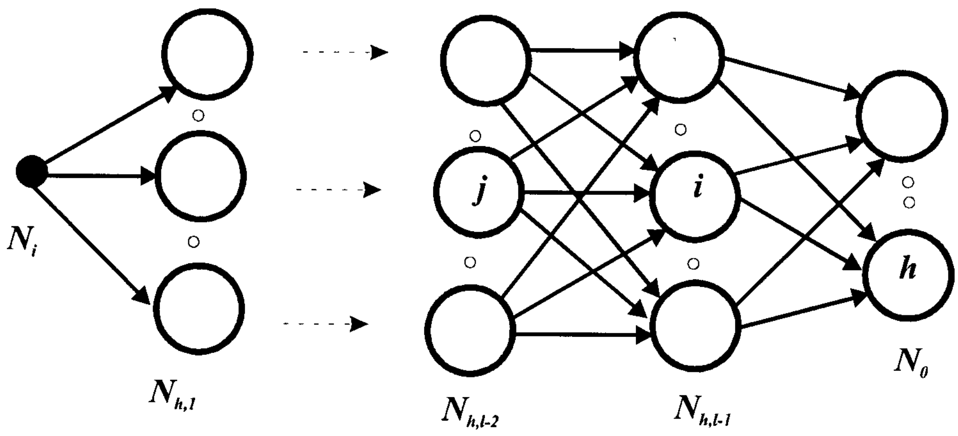


Figure 2.4: A multi-layer network with *l* layers of units.

In Figure 2.4: N_i is the number of input nodes; $N_{h,-}$ the number of units in the given hidden layer; and N_0 the number of units in the output layer.

2.2.2 The Back-Propagation learning algorithm

The back-propagation learning rule applies to multi-layered networks with hidden layers. The central idea behind this method is that the errors for the units of the hidden layer are determined by propagating backwards the errors of the units of the output layer. For the sigmoid activation function, the error signal δ_i^p of the *i*th **hidden unit** (of the *p*th input pattern) is given by:

$$\delta_i^p = F'(i_i^p)\sum_{h=1}^{N_0}\delta_h^p w_{hi} = a_i^p(1-a_i^p)\sum_{h=1}^{N_0}\delta_h^p w_{hi} \tag{2.6}$$

where a_i^p is the current activation of the *i*th neuron (for the *p*th input pattern). $F'(i_i^p)$ is the derivative of the activation function with respect to the effective input to the *i*th neuron of the *p*th pattern, i_i^p. If the unit is an **output unit,** the error signal is given by(for sigmoid activation):

$$\begin{aligned}\delta_i^p &= (d_i^p - a_i^p)F'(i_i^p)\\ &= (d_i^p - a_i^p)a_i^p(1-a_i^p)\end{aligned} \tag{2.7}$$

where d_i^p is the *i*th element of the desired output of the network when input pattern *p* was input to the network. The weight of a connection is adjusted by an amount proportional to the product of an error signal δ_i^p, on the unit *i* receiving the *p*th input pattern and the output of the unit *j* sending this signal along the connection, i.e.:

$$\Delta_p w_{ij} = \gamma\ \delta_i^p a_j^p \tag{2.8}$$

where δ_i^p is given either by (2.7) or (2.6) depending on whether the unit is in the output layer or in the hidden layer respectively. Therefore, in the back-propagation learning rule the error signals of the output units are determined first. Then the error signals for the hidden units are determined recursively in terms of error signals of the units to which they directly connect and the weights of those connections as given in (2.6). In equation (2.6) the hidden units under consideration connect directly to the output units as indicated by the summation of h=1 to N_0, where N_o is the number of units in the output layer. Therefore, if the hidden units under consideration connect to another layer of *m* hidden units above them, then the summation will be taken over h=1 to m [10]. That is to say, the error signals of the hidden layer above, are considered instead of those of the output units as earlier stated. Once the error signal is known, the current weight is adjusted by an amount proportional to the error signal and the activation value from (2.8). The proportionality constant, γ is the learning rate.

The method described above is a basic back-propagation algorithm, which is slow and not very efficient. There are other improved versions of the standard back-propagation method which utilise **adaptive learning rate** and **momentum** principles. The momentum method reduces the risk of getting stuck in shallow minima. To get this effect, the weight changes are set equal to the sum of a fraction of the last weight change, and the new change suggested by the back-propagation rule. In the Matlab (Matlab 4.2c, 1994) implementation, the momentum constant may have a value between 0 and 1. Where (0) means that no momentum is used and (1) means that the new weight change is exactly equal to the old weight change.

2.2.3 The Levenberg-Marquardt training algorithm

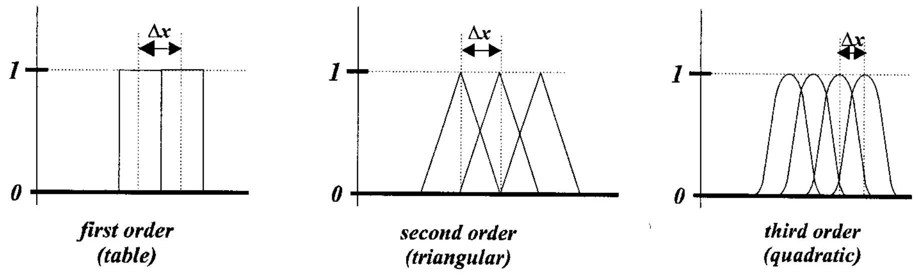
This is a more sophisticated form of back-propagation learning which tries to compromise between gradient descent methods (standard back propagation) and Newton's method. Gradient descent tends to converge very slowly near the solution and will often get stuck in shallow local minima. The Newton direction is often not the same direction as the steepest descent. The Levenberg-Marquardt algorithm alternates between the Newton and descent directions, and ensure that the steps always produce good descents. There is a parameter which causes the update direction to vary between the two directions, and which also controls the step size. In this way, the optimisation method overcomes the problem of getting stuck in shallow local minima and often finds better optimum solutions. The only limitation with this algorithm is that it requires a great deal of memory for large problems. The method is discussed in detail in [13] and [15]. The algorithm has been implemented in Matlab, for a network of up to three hidden layers (Matlab 4.2, 1994).

2.3 B-spline functions

Instead of using sigmoid functions (and other Gaussian functions) as in N-Net structures, a network may use spline functions for its non-linear function. One such type of spline functions is the B-spline function. The idea of using splines is to construct polynomial interpolators on intervals of the input space. B-spline functions are piece-wise polynomials defined on a closed interval. A B-spline of order N consists of polynomials of degree N-1 [4]. For instance, a second order B-spline consists of two piece-wise linear functions (which are polynomials of first the order) forming a triangular function between three points. A third order B-spline will comprise three second order functions fitting four points. Figure 2.5 shows B-spline functions of different order.

2.3.1 One-layer B-spline networks

One-layer B-spline network structures described above are closely related to fuzzy logic structures. Fuzzy logic structures are based on linguistic rules, which give the relation between input-output space. Imprecision in the statements used in the rules is modelled by use of fuzzy sets. It has been shown that if B-spline functions are used as membership functions, a link between fuzzy logic structures and one-layer B-spline network structures can be made [5][16][17].



- ***Δx=cell width (overlapping zone)***

Figure 2.5: B-spline functions of different order

2.3.2 B-spline network design and training

The range of possible values for an input variable is divided into a number of (fuzzy) subsets. In fuzzy logic theory, an input value (within the input range) may belong to more than one subset. Each subset ***i*** is represented by a membership function $\boldsymbol{b_i(x)}$. The value of $\boldsymbol{b_i(x)}$ is a measure of membership of ***x*** to this subset. The B-spline function is the membership function in the case of B-spline networks.

During the design process of a B-spline network, the right choice the of membership function to be used is very important. The reason is that the interpolation behaviour of a B-spline network depends on the order (type) of the spline functions used. Second order B-spline functions support linear interpolation, while higher order B-spline functions support higher order interpolations. Apart from making the right choice of the membership function, the number and position of the function is equally crucial [4]. Knowledge of the form of the function is necessary in order to make a proper choice of the positions of the membership functions. In irregular areas of the function to be approximated, more membership functions are needed to increase accuracy.

A B-spline network structure, may have a single layer or more including hidden layers. Figure 2.6 shows a one-layered B-spline network. The output is determined by the summation of the product of the membership value for a given input value, and the corresponding weight value as follows:

$$y = \sum_i b_{A_i}(x) w_i \tag{2.9}$$

Where b_{A_i} is the membership to a particular cell (overlapping zone of membership functions), w_i is the associated weight and y is the network output.

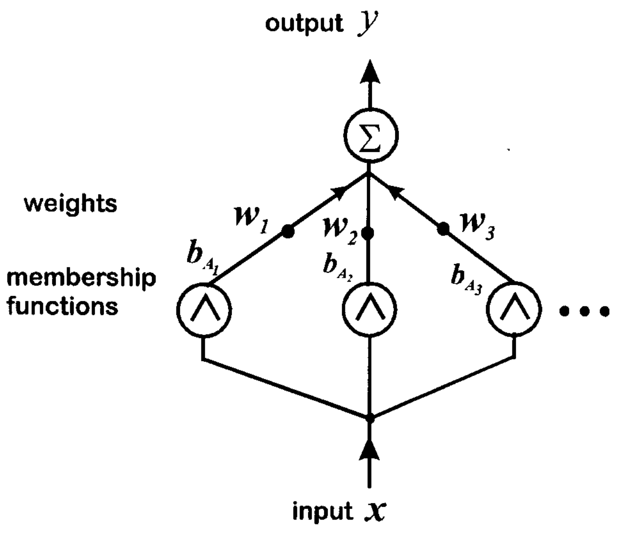


Figure 2.6: A B-spline network (one-layered).

A B-spline network is also trained according to some learning rules. Because of the linear form of (2.9) and also the continuity of B-spline functions, it is possible to use gradient learning rules. Gradient methods minimise a cost function. In the case of feed-forward N-Nets, the objective function is usually taken to be the mean square error (MSE). The derivation of a gradient descent rule for the B-spline network is similar to that of the delta rule. The delta rule forms the basis for the back propagation rule presented in Section 2.2.2. Defining the MSE as:

$$E = \frac{1}{2}\left(d(x) - y(x)\right)^2 \tag{2.10}$$

where *d(x)* is the desired output and *y(x)* is the actual B-spline output. Then differentiating the error E with respect to the weight w_i yields:

$$\frac{\delta E}{\delta w_i} = -\left(d(x) - y(x)\right)b_{A_i}(x) \tag{2.11}$$

Therefore, the weights are adjusted according to the rule:

$$\Delta w = \alpha\left(d(x) - y(x)\right)b_{A_i}(x) \tag{2.12}$$

The B-spline network is therefore trained by presenting a training data set, comprising the inputs and the corresponding (desired) outputs. The total number of weights in a one-layer B-spline network structure is a measure of the memory used. The parameter, α is the learning rate and is analogous to γ in (2.8).

2.4 The concept of optimal control

It has been stated that N-Nets and indeed B-spline networks can be trained to perform tasks. If they are to be applied as direct controllers, then they have to be taught the control task to carry out. This implies that control knowledge must be available. Often, numerical methods are used to generate this data. A numerical method of Differential Dynamic Programming has been found to be suitable for applications such as the Inverted Pendulum [4][5]. A summary of this numerical method is presented in this section.

2.4.1 Differential Dynamic Programming (DDP)

Differential Dynamic Programming is an optimal control numerical method. used to generate optimal control data. An optimal control algorithm steers a discrete process in open loop from an initial state *x(0)* to a final state *x(N)*, while minimising a performance criterion or objective function, ***J*** [18]. Several other optimal control methods are available for solving two point boundary problems. These include the maximum principal of Pontryagin, calculus of variations, and dynamic programming [4]. Dynamic programming just like the maximum principal of Pontryagin, is suitable for problems with limited control space [19][20][21][22] . This makes it ideal for application to the Inverted Pendulum non-linear control problem which has limited control space.

Dynamic programming is a multistage decision process [22], based on the principal of optimality, which is stated by Bellman [19], as: "an optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." However, since the calculation of an optimal control, given non-linear difference equations (of the Inverted Pendulum model) with two-point boundary conditions is not analytically solvable , a direct method , the gradient method may be used. Differential Dynamic Programming (DDP) is a computational method that combines both the gradient method (first order) and dynamic programming [4][18].

In general, the DDP algorithm requires that the discrete time dynamic system is described by difference equations of the form:

$$x(k+1) = f\big(x(k), u(k)\big) \ , \qquad x(0) = x_0 \ , \qquad x(N) = x_N \tag{2.13}$$

where $x(k)$ and $u(k)$ are the complete state vector and control variable at time k respectively. x_0 is the initial state and x_N the final state. The control $u(k)$ is constrained as follows:

$$|u(k)| < |u_{\max}|$$

The performance criterion is the cost function to be minimised and is given by:

$$J = \Phi(x(N)) + \sum_{k=0}^{N-1} g(x(k), u(k)) \tag{2.14}$$

or by:

$$V(x(k), u(k)) = g(x(k), u(k)) + V(x(k+1), u(k+1)) \tag{2.15}$$

where $J = \Phi(x(N))$ represents the desired state condition at time N. The other term, $g(x(k), u(k))$, defines the state and control conditions to be minimised for the time period $k = 0, 1, \cdots, N-1$.

The control problem is to determine the control trajectory $u(k)$ which minimises the cost function given by (2.14). This is an iterative process and a decision about the optimal value of $u(k)$ is made at each stage of the process [22] as shown in Figure 2.7.

The full set of equations showing the rationale behind optimal decisions regarding the control, are given in Appendix B. A summary of the DDP algorithm [4] is given in Appendix C.

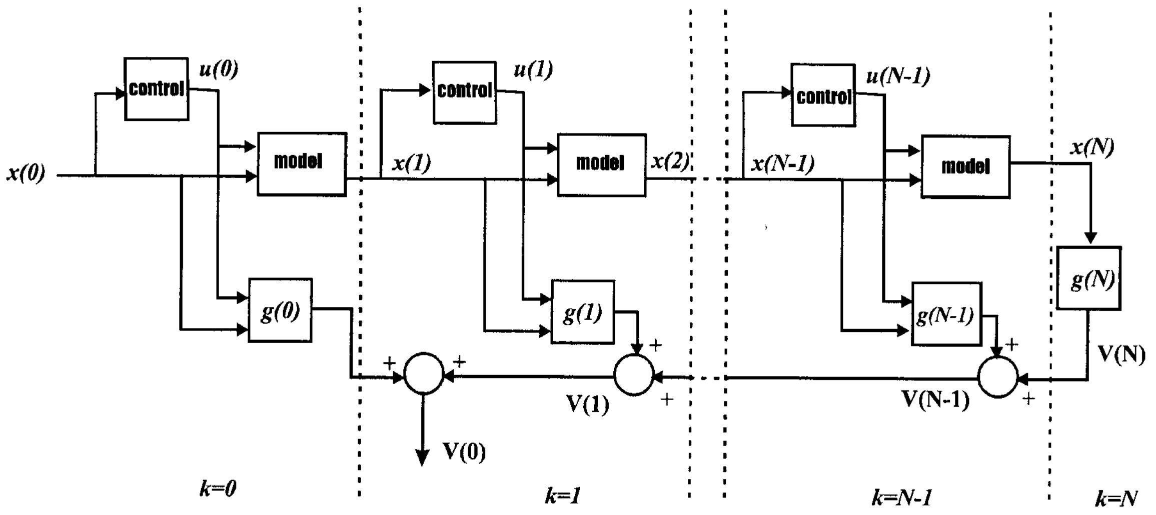


Figure 2.7 Differential Dynamic Programming showing the computation of the cost

2.5 Conclusion

In the last few years a lot of research work has been done in the exploration of AI techniques and methods and its theory is becoming richer. However, the AI theory is yet to be "standardised". The theories that have so far been explored are already finding their practical use in systems and control applications. Given that nodes in the N-Net are computation units, the forward calculation speed of the N-Net must be determined by the number of nodes in the N-Net chain. Thus, the fewer the number of hidden nodes, the faster the calculation speed of the N-Net. The forward calculation speed is important during real time application of N-Nets.

3.0 OS BASED CONTROL OF INVERTED PENDULUM

Operation-space (OS) based control is an arrangement in which different controllers are applied during different states of the system. This is done when one single controller is not able to cover the full state-space of the system. The technique was used by Jongma [4], to combine three different controllers to control the Inverted Pendulum. Before the new controller was designed, the organisation and performance of the existing OS based controller were studied. This was necessary to facilitate the design of a "compatible" controller. Therefore, this chapter presents a conceptual design of the OS based controller and its constituent controllers.

3.1 Classification of the state-space

The state space of the balancing arm can be placed into two categories or zones namely the linear region and the non-linear region [3][4]. This is illustrated in Figure 3.1. For the sake of simplicity the angle of the driving arm is assumed constant. Dividing the state space of the balancing arm into the two zones simplifies the control problem because only one cost function needs to be considered for design of an optimal controller for each region [4]. When the balancing arm is in zone 1 which is basically a linear model, an LQ controller is used to balance the arm. When the LQ cannot balance the arm any more (i.e. $|\varphi| > \pi/3$ rad or $|\theta|>\pi/12$ rad), control must switch to another controller, in this case to the NL controller (zone 2). A non-linear controller was designed by Jongma [4], by teaching an N-Net off-line (see Section 2.5). However, before passing control from the LQ controller to the NL controller, a PD controller is used to steer the driving arm into the downward position needed as the initial state for the NL controller. This is illustrated in Figure 3.2 below (cases 1 to 3).

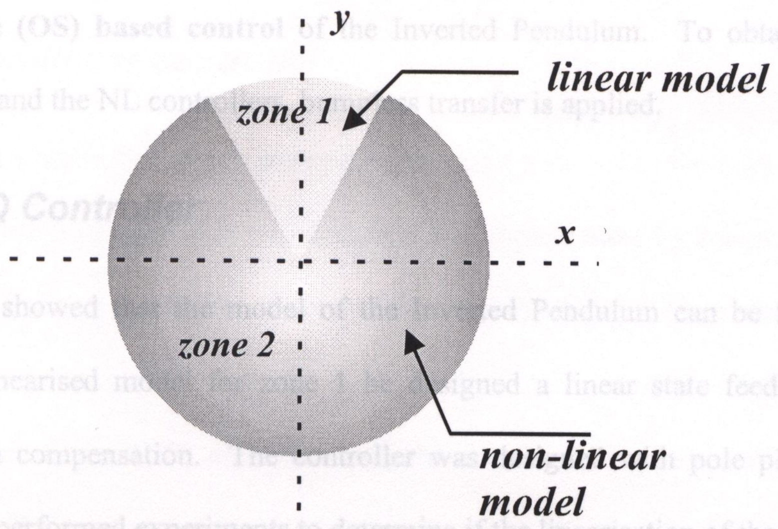


Figure 3.1: The state space of the Inverted Pendulum divided into two zones

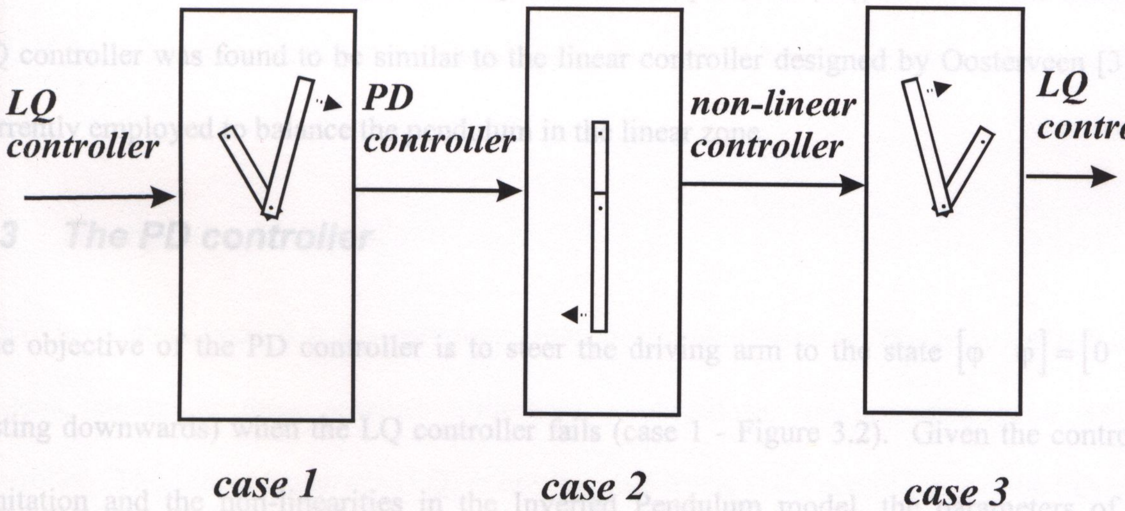


Figure 3.2: Operation-space (OS) based control

When the NL controller has succeeded in bringing the balancing arm up to within the linear zone, control is switched back to the LQ controller (case 3). In this example the balancing arm falls to the right. However, due to symmetrical nature of the Inverted Pendulum model, the same is true for the arm falling to the left except for the changed (inverted) signs in the PD and the NL controllers. Thus the three OS based controllers are combined in this manner to achieve

operation-space (OS) based control of the Inverted Pendulum. To obtain smooth transition between the LQ and the NL controllers, bumpless transfer is applied.

3.2 The LQ Controller

Oosterveen [3], showed that the model of the Inverted Pendulum can be linearised in zone 1. Based on the linearised model for zone 1 he designed a linear state feedback controller with coulomb friction compensation. The controller was designed with pole placement techniques. Jongma [4] also performed experiments to determine if the linearisation of the model in zone 1 was allowed. He found that the Inverted Pendulum did not exhibit significant non-linear behaviour in zone 1. Based on these findings, he designed a Linear quadratic (LQ) controller for zone 1. This LQ controller was found to be similar to the linear controller designed by Oosterveen [3], and is currently employed to balance the pendulum in the linear zone.

3.3 The PD controller

The objective of the PD controller is to steer the driving arm to the state $\begin{bmatrix}\varphi & \dot{\varphi}\end{bmatrix} = \begin{bmatrix}0 & 0\end{bmatrix}$ (i.e. resting downwards) when the LQ controller fails (case 1 - Figure 3.2). Given the control signal limitation and the non-linearities in the Inverted Pendulum model, the parameters of the PD controller were tuned in a series of experiments. The PD controller parameters K_p and K_v were found to be 7 and 0.93 respectively [4].

3.4 The non-linear controller

A non-linear (NL) controller which steers the balancing arm back into zone 1 after failure of the LQ controller (Case 2 - Figure 3.2) was designed and implemented by Jongma [4]. The controller was designed by training a B-Spline structure (third order) off-line. The training data set consisted of non-linear combinations of state and control which were generated by using a Differential Dynamic Programming (DDP) algorithm and the fourth order process model of the Inverted Pendulum (1.1). By training a network structure on the open loop data set obtained from the DDP algorithm, and applying it as a non-linear feedback controller, closed loop control was obtained. The B-spline network is able to compute correct control signals from the given state inputs, but only if the initial state values at the beginning of the control action, are within the specified state space.

3.5 Bumpless transfer

It has been mentioned that the complete controller implemented on the Inverted Pendulum by Jongma, comprises three separate controllers combined in one operation-space (OS) based controller. The Linear Quadratic (LQ) controller, non-linear (NL) controller and the PD controller switch between themselves depending on the state in which the system is. It was established that switching from the NL controller to the LQ controller caused a rough and undesirable transition [4]. Therefore, to obtain a smoother transition between the two controllers bumpless transfer was introduced. A bumpless transfer zone was created between the controllers. In this zone a linear transition from the NL controller to the LQ controller takes place. The switch between the NL controller and the LQ controller takes place when $|\theta| = \pi / 12$, so, the suitable transition zone etween these two controllers was chosen to be $\pi / 18 < \theta < \pi / 9$. The control in this transition one is calculated as follows[4]:

$$u_{\text{trans}} = \frac{u_{\text{non-linear}}(x) - u_{LQ}(x)}{\theta_1 - \theta_2}(\theta - \theta_1) + u_{\text{non-linear}}(x) \tag{3.1}$$

Where

x is the state

$u_{\text{trans}}(x)$ = the control in the transition zone

$u_{\text{non-linear}}(x)$ = the control computed by the NL controller

$u_{\text{LQ}}(x)$ = the control computed by the LQ controller

$\theta_1 = \pi / 18$ and $\theta_2 = \pi / 9$ are the values of θ at the edges of the transition zone

θ = the angular displacement of the balancing arm

3.6 *Implementation of the operation-space based controller*

The OS based controller is implemented in form of software algorithms on a transputer based system. Since the controller is a feedback system, system states must first be sampled before the next control signal can be calculated [23][24]. This may be achieved by having an extremely fast processor or by utilising the principle of parallelism. The latter is a natural solution considering the Inverted Pendulum system in real time, is parallel in nature. The OS based controller was therefore implemented on parallel hardware. The sampling interval is h=0.01s. This Section highlights the available hardware and software and how they are organised to achieve real time parallelism on the Inverted Pendulum setup [2].

3.6.1 Hardware resources

The hardware available comprises a transputer network (T414) which is located in a host PC (486DX). The host PC provides Input / Output (I/O) interface between the user and the system. It takes care of the keyboard, screen handling and filing (disks) functions [2]. The transputer network carries out calculation of the control signals. The transputers communicate with the PC through library function routines. Figure 3.3 shows the way the PC is connected to the transputer network.

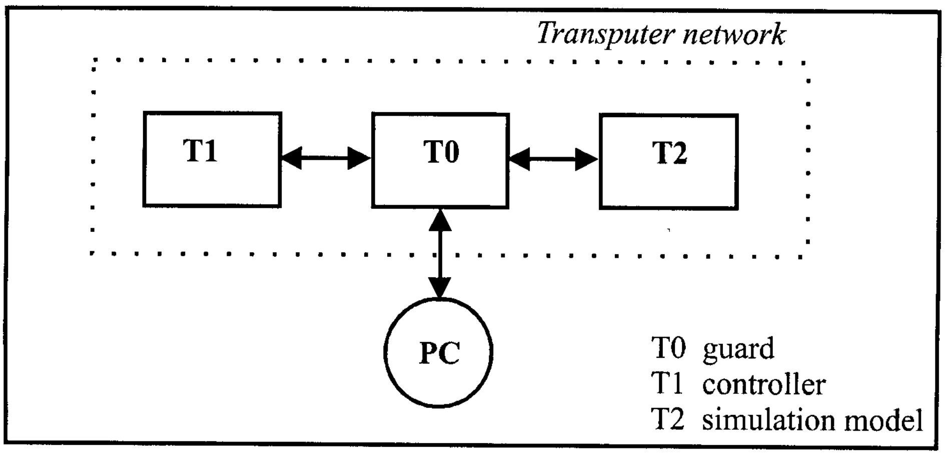


Figure 3.3 Configuration of transputer system on the Inverted Pendulum

3.6.2 Software resources

The software on the host PC has been implemented in Borland C (4.0). The implementation on the transputer system has been done using occam-2 programming language (Occam-2, 1988) and the transputer development system TDS2 (Inmos Ltd, 1988).

3.6.3 Achieving parallelism on the laboratory setup

Due to the high demand for computation speed during real time control, tasks are shared between the host PC and the transputer network. Transputer T0 runs a guard process. Its task is to determine the state of the Inverted Pendulum during each sample interval and to select the appropriate controller (out of the three available controllers in the OS based controller) for the next sample interval. Information of the selected controller, together with the system states are forwarded to Transputer T1, which contains all the controller algorithms. The controller transputer upon receiving data from the guard process, will decide which controller to execute. The appropriate controller algorithm is executed and the next control signal is computed.

Apart from being the host to the transputers, the PC takes care of the user interface as well as presentation of part of the data on the screen (graphs and Inverted Pendulum drawing). In this way, parallelism is achieved as the three processes are running simultaneously (see Figure 3.3). Synchronisation of the processes is achieved by making the guard transputer wait for a value from the PC to mark a new sample interval [3]. This arrangement is shown in Figure 3.4

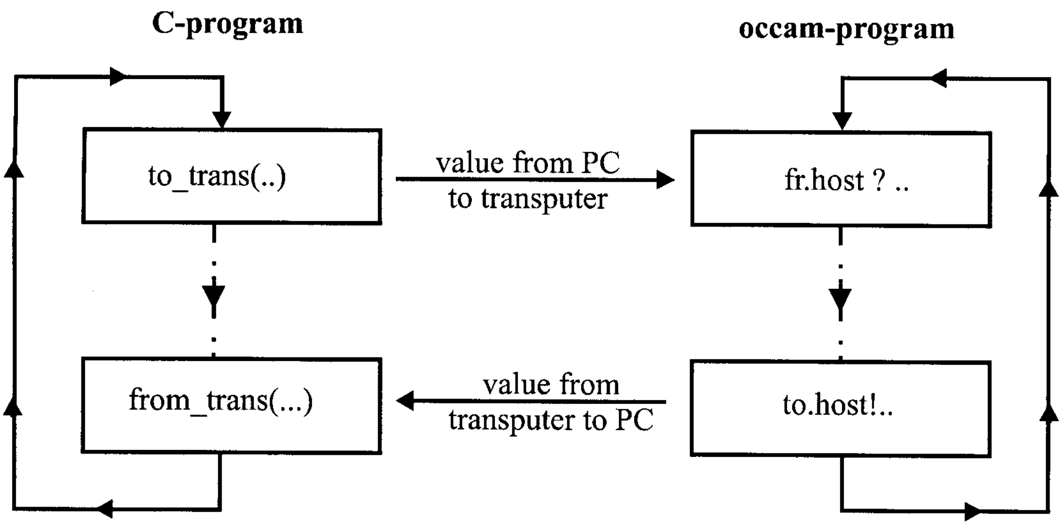


Figure 3.4 Synchronisation of parallel processes on the real Inverted Pendulum set

3.7 Conclusions

Applying DDP to design a NL controller for the whole four-dimensional state space of the Inverted Pendulum is impossible, because controlling the Inverted Pendulum in this state space requires at least two different cost functions (zone 1 and 2). However, when the state space is divided into two or more zones, then optimal controllers can be designed and implemented for each zone. The implication is that to be able to cover the whole state-space, a number of controllers are required. In this respect, the OS based controller covers only a part of the state space. The remaining state space still has no controller(s). Therefore, should the Inverted Pendulum fall in any of these states, the OS based controller fails.

Therefore, more than one controller , is therefore required to control the Inverted Pendulum in the uncovered state space. The prime objective remains the same - to keep the balancing arm in zone 1, the linear region where the LQ controller takes care of balancing the arm upright.

4.0 CONTROLLER DESIGN

In chapter two, the principles of ANN structures were presented. Chapter three gave a brief summary of the OS based controller. In this chapter, an N-Net controller for the Inverted Pendulum is designed. The controller will be applied on the state space which is outside the OS based controllable state space. For this reason, the controller must be compatible with the existing OS based controller, which would continue to handle control of the Inverted Pendulum in the previously known state space.

4.1 Definition of the swing up control task

4.1.1 The downward resting position

Normally, control of the Inverted Pendulum begins while it is in the clamped position. This position corresponds to the state, $x = \begin{bmatrix} \varphi & \dot{\varphi} & \theta & \dot{\theta} \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$. This is also the balancing position. In this state, the OS based controller will control the Inverted Pendulum through the LQ controller, when the clamp is energised and the balancing arm is released. For any other initial state, outside the clamped position, OS based control will result in failure of the controller. For instance when the Inverted Pendulum is in the downward resting position (see Figure 4.1), non of the three controllers in the OS based controller is able to perform a successful control task. The LQ controller was

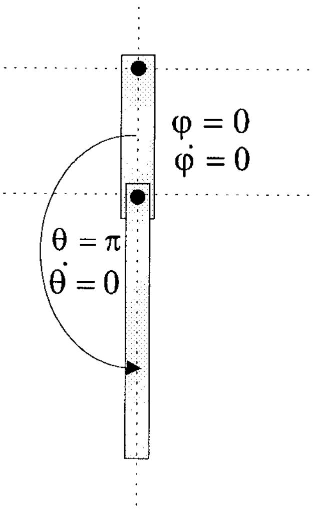


Figure 4.1 The Inverted Pendulum in its stable state - the downward resting position

designed for the linear region, zone 1 and cannot be used in the non-linear zone. As for the NL controller, the downward resting position is not an appropriate initial state for the B-spline mapping function. Therefore, the non-linear controller is not able to steer the pendulum into zone 1. The PD control action would not bring about the necessary initial state for the NL controller. The downward resting position is therefore an unknown initial starting state to the current OS based controller and results in controller failure.

4.1.2 Failure of the OS based controller

In Section 4.1.1 , the downward resting position is shown as a state which causes the OS based controller to fail. However, there are more states which result in the failure of the controller. These states may arise during control itself. That is, disturbance may drive the system into unrecognisable states. If the balancing arm is in zone 1, the LQ controller is able to control and balance it. If the arm is pushed too hard on either side, the LQ controller tries to balance it until $|\varphi| > \pi/3$ rad or $|\theta| > \pi/12$ rad when it is not able to control it any more (zone 2). The arm will then fall off. The PD controller is then engaged and will steer the driving arm to the state$\begin{bmatrix}\varphi & \dot{\varphi}\end{bmatrix}^T = \begin{bmatrix}0 & 0\end{bmatrix}^T$, while the balancing arm is left to fall freely. Once the driving arm is in position or nearly so, the NL controller can now take over control and move the balancing arm (by moving the driving arm) towards zone 1. Here, control is handed back to the LQ controller. During this time, a disturbance may prevent the NL controller from achieving the expected final state. Consequently, the LQ controller cannot take over control, resulting in failure.

After failure, an extra controller is needed to restore the system back into the balancing position. The task of the swing up controller is therefore, to steer or swing the system from the state of failure to any of the desired states - the states controllable by any controller within the OS based controller.

4.2 Swing up strategy

It has been observed that when the Inverted Pendulum is in the downward resting position, at the start of the OS based controller, the result is control failure. In order to prevent this failure, the current OS based controller must be extended to cover the downward resting position in its state space. There are two possibilities of doing this. The first method could be to modify one or more of the three controllers available. The second possibility is to design an extra controller which would be part of the OS based controller. The two possibilities are now explored further.

4.2.1 Modification to existing controllers

This approach would involve modifying either the PD, NL or LQ controllers to include the desired extra state space. The LQ controller cannot be modified for this task because its control action is optimised for the linear region, whereas the downward resting position is in the non-linear region. Also nothing can be done to the PD controller , to perform the needed swing up task. The non-linear controller offers possibilities for this kind of modification. However, that implies designing a new NL controller altogether to replace the existing one. This NL controller could still be a trained N-Net structure or B-spline network, but whose training data set would have to include the downward resting position.

4.2.2 Designing the fourth controller

This option entails keeping the existing controllers in the OS based controller and designing a fourth controller. This controller will handle control of the Inverted Pendulum when it is in the downward resting position, and bring it up to a state controllable by the existing OS based controller. This approach has an advantage over the first option, because it reuses the available resources, namely the existing OS based controller (see Figure 4.2). The swing up problem will therefore be tackled from this angle.

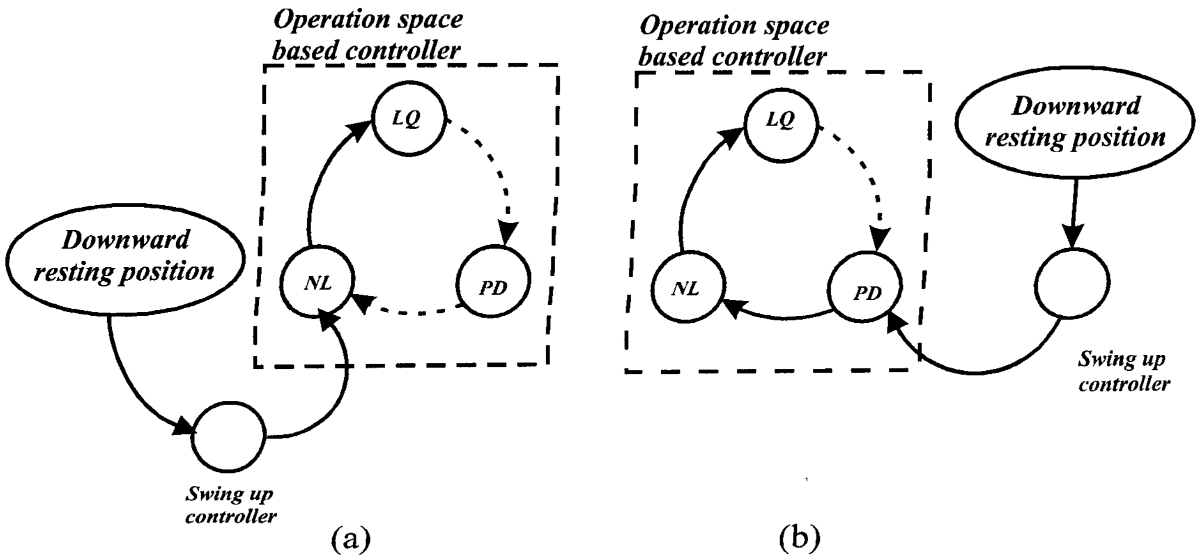

Operation space based controller
LQ
NL
PD
Downward resting position
Swing up controller
(a)
Operation space based controller
LQ
NL
PD
Downward resting position
Swing up controller
(b)

Figure 4.2 Swinging up from the downward resting position (a) NL controller as entry point (b) PD controller as entry point.

f a fourth controller is to be designed, and one which should re-use the existing OS based ontroller, then the controller should be able to drive the system from whatever state to a state controllable by the existing OS based controller. The point of entry into the OS control domain could therefore be through any of the three controllers. Figure 4.2 shows two possibilities, that is, the NL controller and the PD controller as entry points. The third option would be to have the fourth controller take the system straight to the balancing position and allow for LQ control of the Inverted Pendulum.

4.3 Experimenting with combined PD control swing up

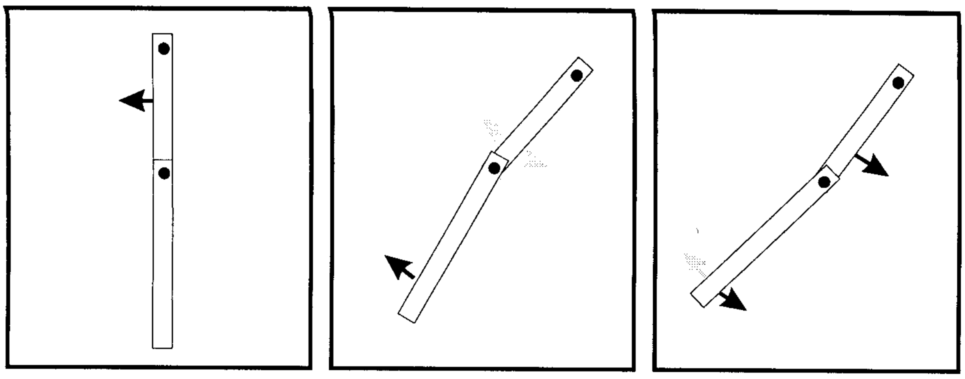
The downward resting position is in the non-linear zone and would therefore require a non-linear controller for a good swing up performance. However, as a preliminary to the NL swing up controller design, some experiments were carried out to swing from the downward resting position by using a combination of PD controllers. The objective was to have an insight into the swing up from rest control problem. First, PD controllers were combined in a time resonance based setup. Then the swing action was transformed into a state dependent one. The PD controllers were carefully tuned until a satisfactory performance was achieved. The design procedure of these controllers is discussed in the Sections that follow.

4.3.1 Time resonance based swing up

It was observed that actuating the driving arm to and forth at some frequency, generated sufficient energy in the balancing arm. Two things are critical and had to be determined experimentally, namely: time at which to swing the driving arm, and how far to swing the driving arm in either direction. Through many trials using video clips of the pendulum in action played back at slow speed, a suitable range of motion of the driving arm was found. Also, the delay time was determined. The delay time is needed before swinging the driving arm towards the opposite direction to increase (and NOT damp) the energy in the balancing arm. PD controllers whose parameters had to be tuned were used for steering the driving arm back and forth. By repeating the back and forth movements of the driving arm a few times, the balancing arm gains enough energy to enter the OS based controller cycle successfully. To re-use the available OS based controller, the best 'entry' point was found to be the NL controller (see Figure 4.2). This is attributed to fact that the PD controller observes only the states of the driving arm while the NL controller requires all the states of the pendulum to compute the value of the control. In addition, the NL controller requires specific initial states which are known (see Section 4.4.4). This means once the initial states of the NL controller are achieved, the transfer of control is certain. As for the PD controller, its ability to hand over control to the NL controller within the OS based controller depends on state when it is invoked. This state(s) is not well defined, except that it is the state immediately after failure of the LQ controller, and so may not be achieved all the time by a different controller. The arrangements shown in Figure 4.2 are open loop systems with regard to the system states and are therefore expected to be very sensitive to disturbances and parameter variations.

4.3.2 State dependent swing up

Having the swing action time based implies that re-tuning of the parameters in the 'controller' is necessary every time any model parameter varies. There is completely no robustness. A better method is to make the swing actions dependent only on the system state. Therefore, using the insight information gathered from the time based resonance swing action, changes were made to enable the driving arm switch directions at intervals determined by the system state occurring at given instances. In this way, the combined PD action incorporates feedback in the system. For instance, it was observed that the best time to swing the driving arm in the reverse direction is when the balancing arm is just about to reverse its direction. This is when it is just coming to a halt as shown in Figure 4.3.



(a) swinging from downward stable position (b) Driving arm stationary, balancing arm still swinging (c) balancing arm about to reverse direction

Figure 4.3: Reversing the driving arm swing direction during swing up.

Again two separate complimentary PD controllers are used to steer the driving arm to and forth, and a third PD controller to place the driving arm in the right position for NL control. The swing motions were tuned until it was possible to swing the balancing arm in two swings only. The best reverse swing position was found to be around $76^o < |\varphi| < 120^o|$ and $66^o < |\theta| < 102^o|$.

In the state dependent swing up algorithm, one PD controller (LeftPD) swings the driving arm to the left. The target position is $\varphi = -1.5\text{rad} \ (\approx 86^o)$. Once in position, the controller 'waits' for the balancing arm also to be in position, in this case to come to rest or nearly so. At this point another PD controller (RightPD) moves the driving arm to the right. The target position is now $\varphi = -1.2\text{rad} \ (\approx 69^o)$. When the driving arm is approximately in this position, the third PD controller (HomePD), steers the driving arm to the downward position and control can now enter the OS based controller. The point of entry into the cycle is the NL controller. In both the LeftPD and RightPD controllers, the appropriate speed with which to swing in either direction and how far to swing, were determined through many trials. The parameters for both PD controllers were found to be $K_p=20$ and $K_v=-1.2$, and for the third PD controller, $K_p=10$ and $K_v=-1.8$.

An algorithm which combines the Left, Right and Home PD controllers discussed above was implemented on the laboratory real time setup. The PD-swing-up controller was able to agitate the system from rest into a state controllable by the OS based controller. This is demonstrated by the graphical representation of the real time data showing the operation of the designed controller which is shown in Figure 6.6.

4.3.4 Conclusions

It has been shown that it is possible to swing the balancing arm back into the OS based controllable state in just two swings, given the stationary downward resting position as the initial state. A combination of three carefully tuned PD controllers is sufficient to perform this task.

The simple swing up controller consisting of two PD controllers is not robust and only works well for a specific initial state, namely with the balancing arm completely (or nearly so) motionless in the stable downward position.

Having established the fact that swinging up the balancing arm (at least from the stable resting position) and balancing it within the existing space based controller is achievable, the next step is to investigate how an NL controller will perform.

4.4 Generation of non-linear control knowledge

The first step in the design of an N-Net structure is to make the control knowledge available. The method of supervised learning (Figure 4.4) will be used in training the N-Net. Therefore, training data samples must consist of sets of input and corresponding target values. The inputs to the N-Net will be the system states, and the output, the control signal. In this way, the N-Net, once it has been trained, can be used to compute a set of control signals which will steer the system from a known initial state to a desired final state. The optimal control numerical method of Differential Dynamic Programming is used to generate the needed control - state data for swinging the Inverted Pendulum from the downward resting position. This method was discussed in Section 2.4. This Section presents the adaptation of this method to the "swing up from rest" control problem.

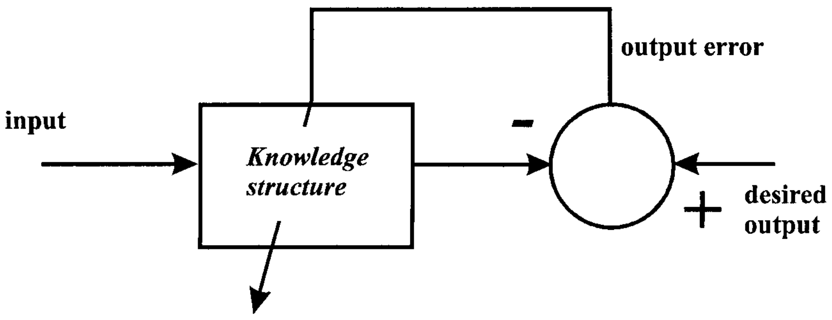


Figure 4.4 Supervised training of knowledge structure

4.4.1 Applying Differential Dynamic Programming (DDP)

To complete the implementation of the DDP algorithm, partial differential equations given in Appendix B are solved for the swing up task of the Inverted Pendulum. The solutions to the partial differential equations are then substituted in the DDP algorithm given in Appendix C. The process requires a discrete model of the Inverted Pendulum and a cost function. Once the DDP algorithm is implemented, the initial state $x(0)$ and the final state $x(N)$ will be needed for the computation of the actual control trajectory (s). The rest of this Section discusses the four items mentioned.

4.4.2 The discrete model of the Inverted Pendulum

Given the van Luenen continuous time model of the Inverted Pendulum (1.1), a numerical integration method is needed to form a discrete model. The corrected version of (1.1) is (D1). Given that the design of the N-Net controller is model-based (i.e. for generation of optimal control knowledge), then an accurate model of the Inverted Pendulum is essential. Therefore, the available continuous time model (1.1) was tested in a step response simulation to check its validity. The results showed an error in the model. The model response showed oscillations of the balancing arm in θ and $\dot{\theta}$, to a zero input control signal while the Inverted Pendulum is in its stable downward resting position (see Figure 4.5). The error was corrected by inspecting the terms in (1.1). The cosine terms in the gravitational term, G of (1.1) were replaced with sine functions. This correction means that the gravitational effect on the balancing arm will be greatest when it is lying flash with the horizontal, i.e. $\left(\theta = \frac{\pi}{2} \text{ or } \theta = -\frac{3\pi}{2}\right)$ as expected. The response of the corrected model is included in Figure 4.5.

After validating the model, it was now discretized by applying two different numerical methods, namely, Euler and Runge-Kutta numerical integration methods. Step response simulation results of the two discrete models were compared against each other as well as against the continuous model. It was observed that for small displacements, the responses of the two discrete models did not differ much. For large displacements, the first order Euler model response deteriorated while that of the second order Runge-Kutta model remained accurate.

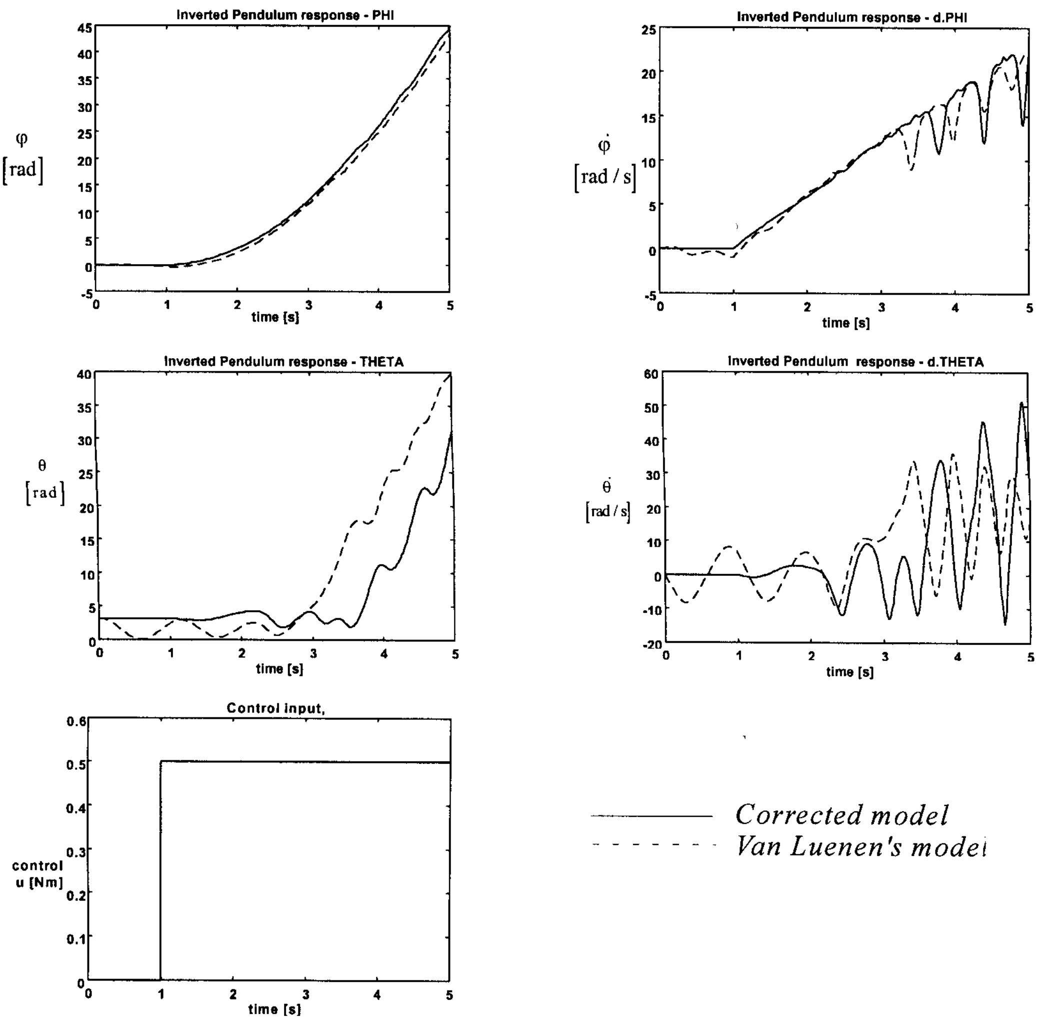


Figure 4.5 Step response - van Luenen's model vs. corrected model

The implication of this result is that the Runge-Kutta model allows for a greater step size h, while maintaining model accuracy. This is an advantage as it would increase the speed of the controller in real time by having a larger sample interval. However, the hardware on the real Inverted Pendulum setup would not support a greater step size (*h* > *0.02s*). Besides, all the controller algorithms were designed and implemented for a step size of *h* = *0.01*s which could not be changed as it would entail re-designing most of the controllers already in existence. Therefore, the simpler Euler discrete model was chosen to be used in the design of the DDP algorithm, as it reduces the complexity of numerical computations in the algorithm while maintaining the same accuracy as the more complex Runge-Kutta model at the shorter sample time of $h = 0.01$s. Therefore, the step size of $h = 0.01s$ was maintained. Figure 4.6 shows the step response of both discrete models and the continuous model. Details of the derivations of both discrete models are given in Appendix D.

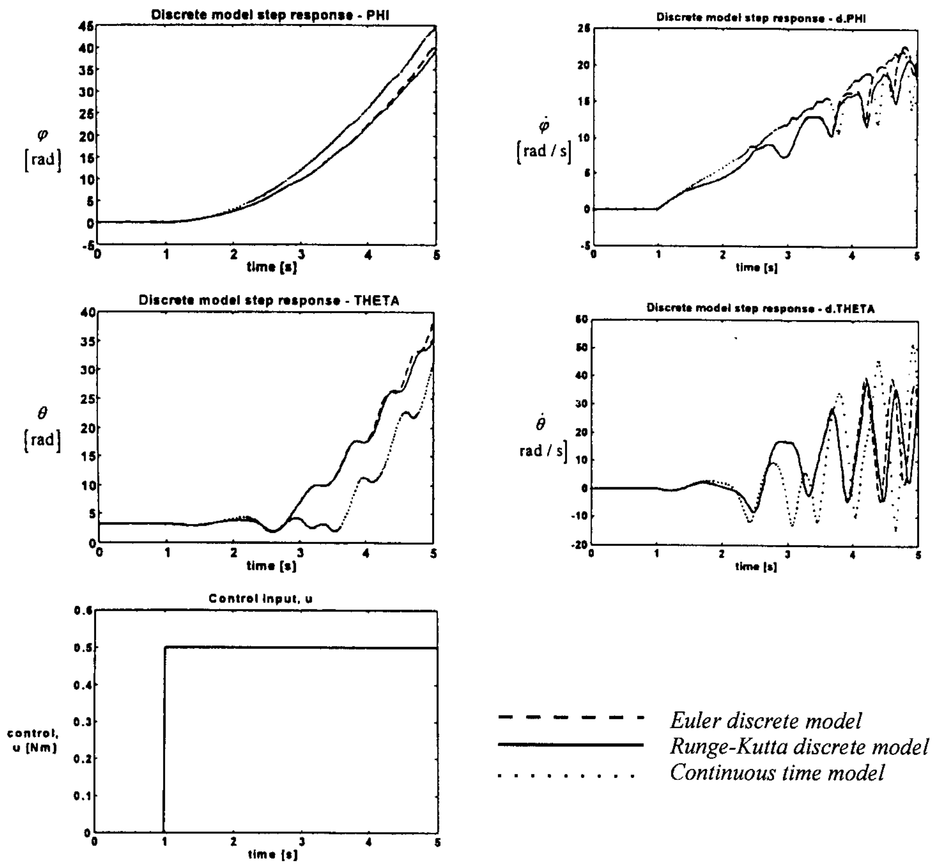


Figure 4.6 Discrete model step response

4.4.3 Choice of the cost function

There are no guide lines available regarding the choice of the objective or generalised cost functions. The procedure is primarily one of trial and error. However, the nature of the general cost functions (2.14) and (2.15) suggests a few considerations that are of help in the process of searching for a cost function. The equations are summations of two terms $\Phi(x(N))$ and $\sum_{k=0}^{N-1} g(x(k),u(k))$. The former term represents the cost associated with the desired state condition at time N. The latter term, $g(x(k),u(k))$, defines the state and control conditions to be minimised for the time period $k = 0,1,\cdots,N-1$. Given that the Inverted Pendulum is to be steered from a state of no energy (downward resting position) to a final state of kinetic energy (entry point into the OS based controller), the cost may be a function of an inverse energy function so that when minimised, the system energy is maximised. The second possibility is that since the final state of the swing up action is what matters, namely, the entry point into the OS based controller cycle, the intermediate terms of the cost given by $g(x(k),u(k))$ can be left out. In such a case, the cost function will be a function of only the four final states (i.e. from $\Phi(x(N))$). The second approach was used in the exploration of the cost function. Thus, it was decided to search for a cost function of the general form:

$$J = Q_1\varphi^2(N) + Q_2\dot{\varphi}^2(N) + Q_3\theta^2(N) + Q_4\dot{\theta}^2(N) \tag{4.1}$$

where the parameters Q_1 to Q_4 represent the sensitivity of each state during the process of computing the cost. The parameters Q_1 to Q_4 then, had to be determined through many trials.

After many experiments, a cost function to steer the Inverted Pendulum in open loop from the downward resting position to the desired final state, would not be obtained. It was observed that with a single cost function of the form (4.1), the driving arm went only in one direction, and did not swing back in reverse, an action which is necessary to generate the necessary energy in the balancing arm as seen in the combined PD swing up control (see Section 4.3). The swing up task was then split into two sub tasks, consequently leading to two separate cost functions. The first cost function steers the system to some intermediate state as quickly as possible. The objective is to increase both kinetic and potential energy in the system. The second cost function steers the system from the final state of the first cost function, to a state good enough for the Inverted Pendulum to enter OS based control. The swing up control task therefore, required two cost functions, one for each sub-trajectory.

After a great deal of trials, the two cost functions were chosen to be:

stage I: $$J_1 = Q_1\left(\varphi(N) - 3\pi\right)^2 + Q_2\dot{\varphi}^2(N) + Q_3\theta^2(N) + Q_4\dot{\theta}^2(N) \qquad (4.2)$$

stage II: $$J_2 = Q_5\left(\varphi(N) + 2\pi\right)^2 + Q_6\dot{\varphi}^2(N) + Q_7\left(\theta(N) - \frac{\pi}{2}\right)^2 + Q_8\dot{\theta}^2(N) \qquad (4.3)$$

The constants $Q_1,\cdots,Q_8$ determine the ratios between the state parameters at the final state for each stage. The constants for the final stage (stage II) are, therefore, more critical as the final state of the second trajectory must fall within specific values in order to successfully enter the OS based control cycle. The values of the constants were adjusted until the final states given in Table 4.3 were achieved. Because there are no guide lines on how to get or arrive at accepted values, principles of sensitivity analysis on the parameters were employed [25]. The final constant values are given in Table 4.1. It was found to be necessary to include the π terms because they determined the total increase or decrease in a state with respect to the other states during the total computation time , *N*.

Table 4.1 Constants for the DDP cost functions J_1 and J_2

Constant Variable	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
Value	0.01	3.0	0.002	0.0005	0.2	90	0.4	0.02

4.4.4 Determining the initial and final states

The initial state of the controller is well defined. It is the downward resting position of the Inverted Pendulum given in Table 4.2. The DDP algorithm must therefore, steer the system from this state and to a desired final state. The state $\theta = \pi$ could also be taken to be $\theta = -\pi$, in which case the swing direction would be reversed.

Table 4.2 The downward resting position

φ (rad)	$\dot{\varphi}$ (rad/s)	θ (rad)	$\dot{\theta}$ (rad/s)
0	0	π	0

The final state of the DDP computation depends on the entry point into the OS based control cycle. In Figure 4.2 two possible entry points are shown. The NL controller was chosen as the entry point due to reasons explained in Section 4.3.1. This implies that the final state of the swing up controller must be within the allowed initial states of the NL controller. Table 4.3 shows the allowed initial state ranges [4]. Therefore, the final states of the DDP trajectories must be within these values or at least very close to them.

Table 4.3 The desired final states of the swing up from rest controller

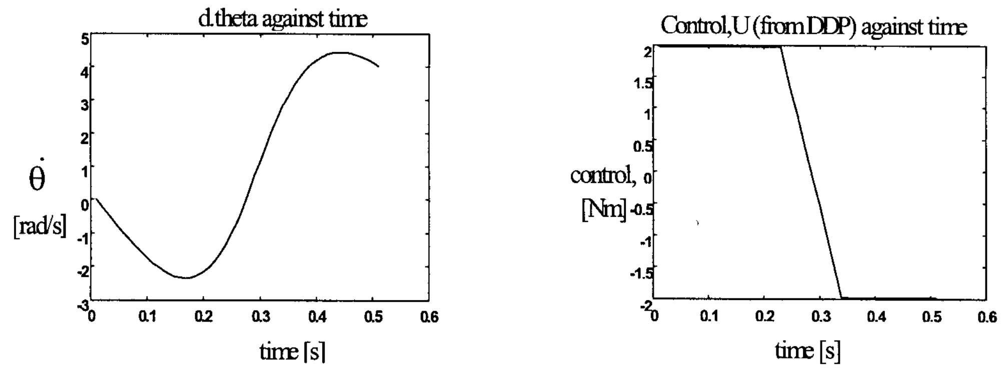
φ (rad)	$\dot{\varphi}$ (rad/s)	θ (rad)	$\dot{\theta}$ (rad/s)
$-\pi/36$ to $\pi/36$	-0.5 to 0.5	$-\pi$	-11 to -16

Since there are two stages in the swing up operation, the final state of the first trajectory was used as the initial state for the second trajectory.

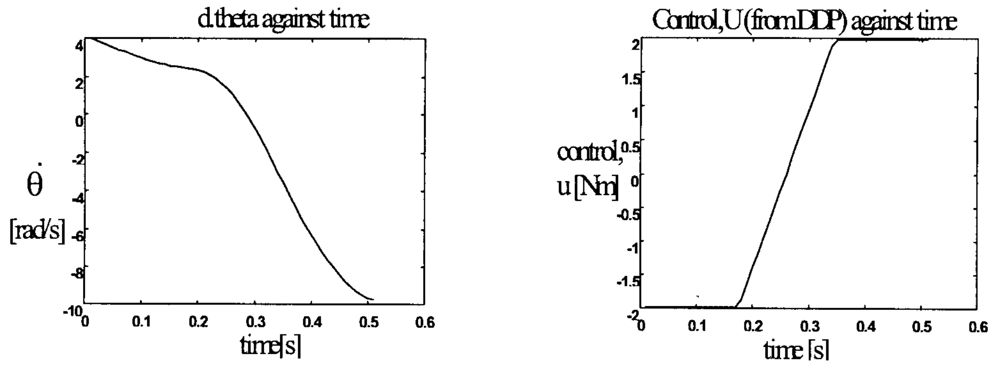
4.4.5 Results from the DDP algorithm

Given the cost functions (4.2) and (4.3), and the initial states in Table 4.2, a DDP algorithm was implemented on a PC compiler. The iterative equations are derived from the fourth-order model which was discretised in Section 4.4.2. Given that the cost functions are functions of only the final states, the optimisation problem transforms into a minimum time steps, N problem. That is, an optimal trajectory becomes one for which the minimum allowed cost is reached with the smallest number of time steps, N. The binary search method in Appendix E was used to find the minimum time.

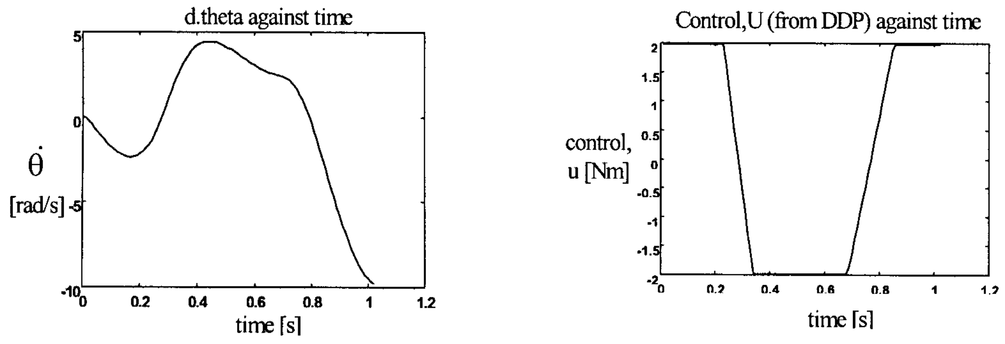
The DDP computer program calculates and stores the control trajectories. The computations take about 4 hours on a Pentium 133 MHz, 32MB PC using the Visual Basic 4.0 compiler. If more initial states are to be considered, the computation time would be much higher. Examples of the two sets of trajectories are given in Figure 4.7. The two sets of data sets are collected in one file which makes up the training data.



(a) Stage I



(b) Stage II



(c) Combined trajectory data

Figure 4.7 Sample results of the DDP algorithm

4.5 *Neural network structure design and training*

Given the control data generated in Section 4.4, an N-Net structure will now be designed. Designing an N-Net structure entails choosing the number of hidden layers, the number of neurons in each layer and the type of learning algorithm to use. This Section presents the design procedure for the N-Net structure to be applied as a non-linear swing up controller.

4.5.1 Hidden nodes

In general, the complexity of the function to be learned determines the number of hidden layers in an N-Net structure. A more complex functions requires more hidden layers. However, only one hidden layer is sufficient to approximate any function with finitely many discontinuities, provided that the activation functions of the hidden units are non-linear (Section 2.2). Therefore, given that the trajectories to be learned are fairly smooth (no discontinuities), one hidden layer will be considered in the design of the N-Net. A non-linear activation function will be chosen for the hidden units. Having only one hidden layer improves the forward calculation speed of the N-Net since the chain of computations is made shorter (see Section 2.2).

Having chosen the number of hidden layers to be one, the next thing to be determined is the number of hidden nodes in the layer. The number of hidden nodes, just like the number of hidden layers, is also dictated by the complexity of the function to be learned. There are no guide lines regarding the choice of the number of hidden nodes, although some rules of thumb may exist. To arrive at an appropriate number of hidden nodes, simulation experiments were carried out. The number of hidden nodes was varied between six (6) and twenty (20) and their effects on the final Mean Squared Error (MSE) observed. Training was done for a fixed number of training epochs

(1,500) with the Standard Back Propagation learning rule. The result is shown in Table 4.4. It was observed that the final MSE depends on the number of hidden nodes. A larger number of hidden nodes leads to a small final MSE. And as Figure 4.8 shows, the final MSE decreases rapidly when the number of hidden nodes exceeds 10. It is desired to have a small MSE because it shows how much the N-Net has learned to map the function represented by the training data.

Table 4.4 The effect of the number of hidden nodes on the final MSE

No. of hidden Nodes	Final MSE
6	0.4258
8	0.2610
9	0.2538
10	0.2081
14	0.1768
20	0.1344

* The Standard Back Propagation Learning Rule was used throughout.
* Number of training epochs fixed at 1,500.

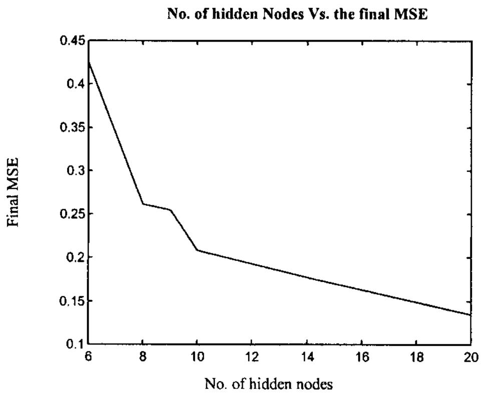


Figure 4.8 Plot of No. of hidden nodes against the final MSE

Having more hidden nodes means more computations which leads to a slow forward calculation speed of the N-Net structure. In other words, there is a trade off between producing a small MSE by having a large number of hidden units resulting in a slow forward calculation speed, and having fewer hidden neurons which improves the forward calculation speed, but results in a high final MSE. The objective is to have a small MSE, but at the same time return sufficient forward calculation speed. The forward calculation speed of the N-Net is of critical importance since it is to perform computations in real-time. Therefore, it will be endeavoured to keep the number of hidden nodes as small as possible. Since the final MSE also depends on the effectiveness of the training algorithm, further experiments with various learning rules were performed before arriving at the actual number of hidden nodes needed.

4.5.2 Training algorithms

A number of learning rules for N-Net training exist. To investigate the performance of each learning rule, an N-Net structure with nine hidden nodes was trained using four different learning algorithms. The result is given in Table 4.5.

Table 4.5 Convergence speed of learning rules and their effect on the final MSE.

Learning Algorithm	Training Epochs	Final MSE
Standard Back Propagation	2500	0.2407
Back Propagation with Momentum	2500	0.1646
Fast Back Propagation	2500	0.0802
Levenberg-Marquardt Optimisation	850	0.0003
* The number of hidden layers is constant at S = 9		

It was observed that the different learning algorithms have different convergence speeds. However, a more critical factor observed is the minimum MSE each algorithm is able to converge to. Some learning algorithms failed to reduce the final MSE beyond certain values (e.g. the back-propagation method). This is because such algorithms get trapped in minimas which are not necessarily the global minima in the solution space. For instance, Table 4.5 shows that the standard back propagation method is insufficient to obtain a low MSE, and the Levenberg-

Marquardt Optimisation method produces the lowest MSE. Figure 4.9 shows an example of the

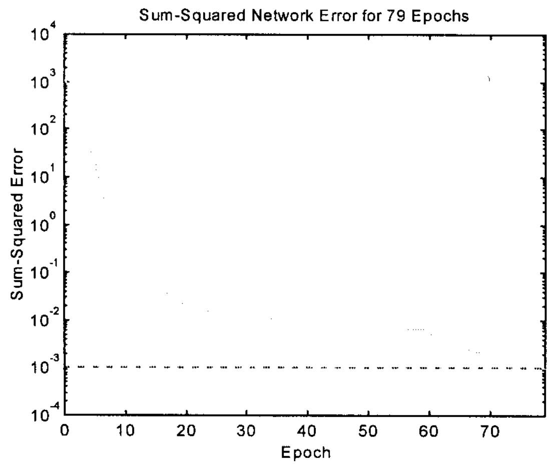
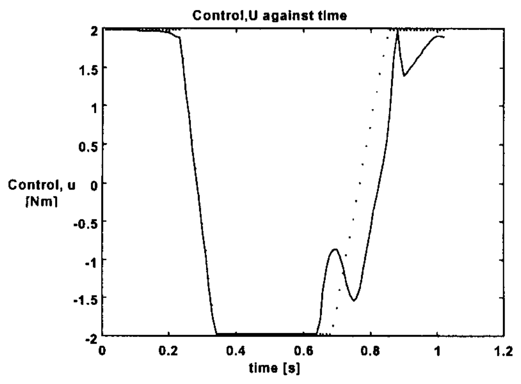


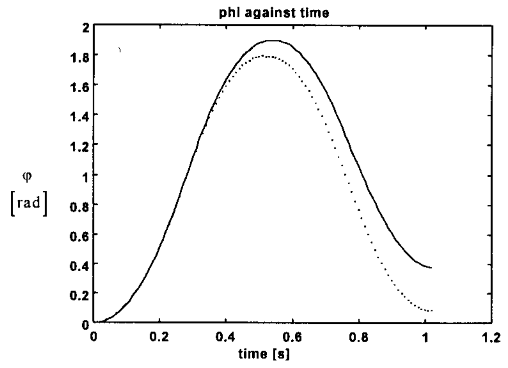
Figure 4.9 A sample plot of the MSE against time during training (Levenberg-Marquardt Optimisation method).

plot of the MSE against time for the entire training period.

4.5.3 Performance evaluation of N-Net structures

The results in Tables 4.4 and 4.5 give the network performance only in terms of the convergence speed during the learning process and the minimum final MSE achieved. However, the real performance of a network lies in its interpolation capability, which is its ability to reproduce the function it is trained to map. A very small final MSE does not necessarily guarantee the best network performance. The trained N-Net structure must be tested on the validation data to check whether it accurately maps the desired function. It is possible to over-train an N-Net structure resulting in a poor network performance. It was observed that having more than ten (10) hidden nodes, easily resulted in over-trained networks. An example of an over-trained N-Net structure is given in Figure 4.10. In the Figure, the control trajectory and the trajectory of the displacement of the driving arm, φ produced by the N-Net are compared with the desired trajectories. It is observed that the final value of φ deviates from the desired value.





———— Neural network output
· · · · · · · · · · · Desired output

Figure 4.10 An example of an over-trained N-Net (30 hidden layers, Levenberg-Marquardt Optimisation method, Final MSE < 0.00001)

The Levenberg-Marquardt Optimisation method shows that it's an effective training algorithm which leads to small MSEs. This ability can be used to reduce the number of hidden nodes resulting in small and efficient but accurately trained N-Net structures. That is, by using this training algorithm, an N-Net structure with a small number of hidden units can still be trained to a small MSE. The small number of hidden units is compensated by the effectiveness of the Levenberg-Marquardt training algorithm to overcome the problem of local minima during the search for the global minima in the solution space. Therefore, the number of hidden neurons was chosen to be eight.

Thus the final N-Net structure comprises four input neurons, eight hidden neurons in one layer and one output unit. The Hyperbolic Tangent Sigmoid transfer function ($a_i = F_i(i_i) = \frac{2}{1+e^{-2i_i}} - 1$)

was chosen as the activation function for the hidden units. The output unit had a linear activation function while no weight adjustments are carried out in the input neurons. The topology of this network is shown in Figure 4.11. This N-Net structure was trained (supervised learning) on the data obtained from the DDP algorithm using the Levenberg-Marquardt Optimisation method. The method of "early stopping" was used to avoid over-training of the network structure. Early stopping refers to the terminating of training to prevent the network structure from learning the peculiarities of the training samples which leads to loss of generalisation. The interpolation behaviour of the trained network is shown in Figure 4.12.

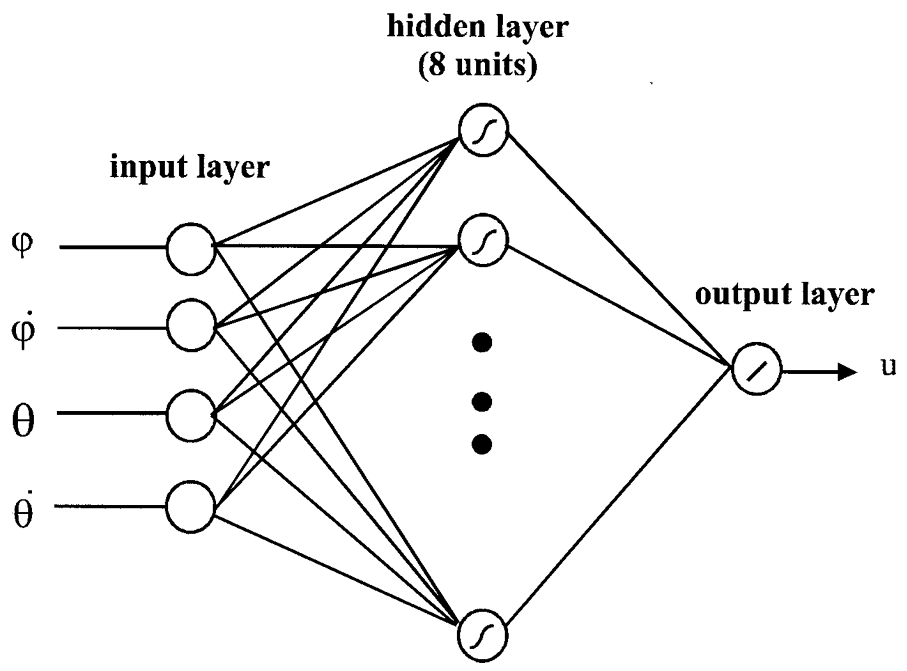


Figure 4.11 Topology of the N-Net to be trained as a controller

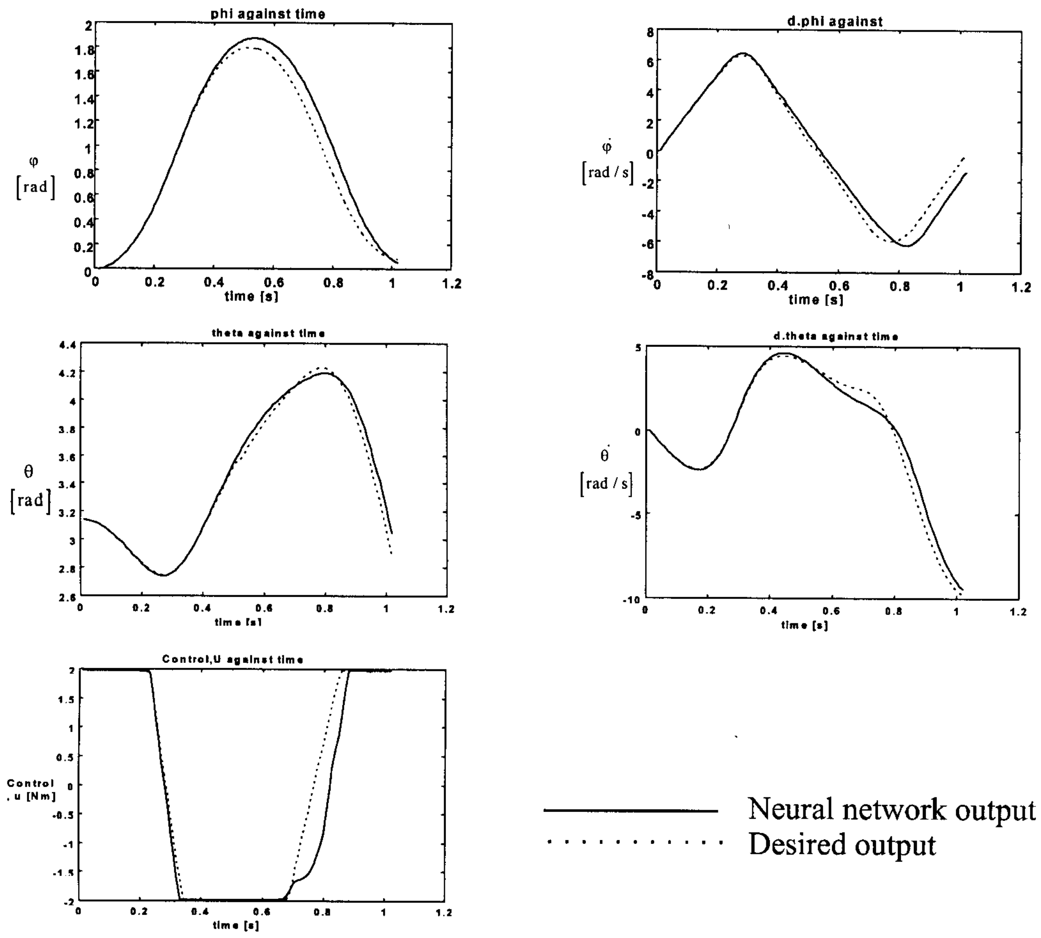


Figure 4.12 Interpolation behaviour of the N-Net to be implemented as the swing-up non-linear controller.

5.0 PRACTICAL IMPLEMENTATION OF THE CONTROLLER

In Section 4, an N-Net structure to perform a swing up from rest of the Inverted Pendulum was designed. The training data comprised control knowledge obtained from a DDP algorithm. The N-Net stores information in form of the weight matrices, network topology and the activation functions used in the network units. Given these three parameters, the problem of retrieving knowledge from the N-Net reduces to a mere computation task. In this Section, the N-Net designed in Section 4, is implemented as a controller on the real Inverted Pendulum setup. The Section also presents the implementation of the combine PD swing up controller designed earlier.

5.1 Implementation of the combined PD swing up controller

The PD swing up controller is a combination of three PD controllers, namely the LeftPD, RightPD and HomePD controllers. Therefore, the swing up controller algorithm is made up of three procedures. Each PD controller is invoked by calling the appropriate procedure in the main controller loop. The controller selection is once again based on the state of the system (Section 4.2.2).

5.2 Implementation the N-Net as a controller

The N-Net controller was implemented in software as an algorithm. In Section 4.2, it was stated that the fourth controller would be designed in a manner that re-uses the already running OS based controller. The controller would only be applied when the system is in the downward resting position. After steering the Inverted Pendulum from this position, the final state must be appropriate for the B-spline NL controller. From this point on, the system has entered the OS based controller cycle, until such a time that the system is back in the downward resting position.

Given the transputer network arrangement highlighted in Section 3.6, adding the N-Net controller algorithm entails defining a new process (procedure). Thus, in the controller transputer, T1, another process is defined. The process is the N-Net controller and it will be called in the main loop. The guard process will select this controller when the system state is in the downward resting position. The information of the selected controller and the system states are passed on to the controller transputer as before.

During the initialisation of the programs, the weight matrices of the N-Net controller are loaded into memory. When the N-Net controller is called in the main loop, the control value is computed by processing the four inputs (system states) through the network, the result being the activation of the output neuron (Chapter 2). This computed value of the control is applied on the system (driving arm) during the next sample interval. At the end of the sample duration, new system states are read and again used to compute the next control value (see Figure 5.1). The process should continue until the Inverted Pendulum enters a state known to the B-spline NL controller. At this point, the guard process will select the B-spline NL controller and the system is back in the OS based controller cycle. In this way, the N-Net controller becomes the fourth controller in the new OS based controller.

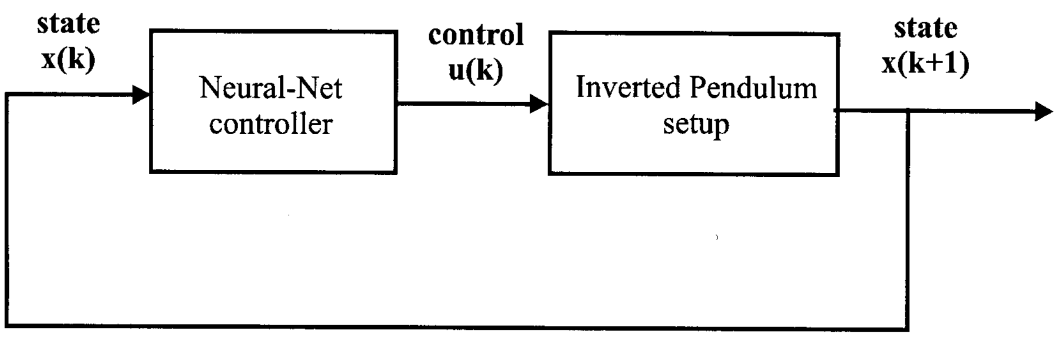


Figure 5.1 Applying the N-Net structure as a controller

A number of errors were experienced during implementation of the N-Net controller. Firstly, it was observed that shortly after commencement of N-Net control, a transputer error developed. The error was pursued, and it was later found out that the error was caused by large numbers which resulted from the evaluation of the exponential function present in the activation function of the hidden units. The Hyperbolic Tangent Sigmoid transfer function, $a_i = \frac{2}{1+e^{-2i_i}} - 1$, was used as the activation function for the hidden units in the N-Net. Large numbers resulting from the computation of the effective inputs, i to the hidden nodes (products of the weights and the outputs of the units in the input layer), may result in very large values when applied as powers to the exponential function. The transputer overflow error is generated when the value of the exponential exceeds the largest number which can be stored by the transputer. The largest number the T414 transputers can store is $\exp(308)$. Therefore, any powers larger than 308 will result in overflow errors. Similarly, all results obtained from arithmetic operations (especially multiplication) which exceed $\exp(308)$, will cause overflow errors.

To try and go round this hardware limitation, the levels of activation for units where exponential values cause overflow errors, were all approximated to either +1 or -1. Obviously, This action has a negative effect on the interpolation behaviour of the N-Net. But it allows the neural controller to compute control values. In simulations of the N-Net controller, the largest number on the Pentium PC was exp(709) which is much larger.

The second problem that was experienced was the hardware limitation on the displacement of the driving arm. It was discovered that a hardware mechanical brake has been set at $\varphi = \frac{\pi}{2}$. This means that in the present form of the Inverted Pendulum setup, displacement of the driving arm is not allowed to exceed this value. As a safety measure, each time $\varphi = 80\ \text{deg}$, the calculated value of the motor torque signal is blocked so that no output from the amplifier is fed to the DC motor. This was done to prevent damage on the setup which can result if the driving arm was allowed to hit the mechanical brake at high speed. But this physical limitation on φ affected the operation of the N-Net controller which requires a higher displacement of φ. This problem is discussed further in the following chapter.

6.0 SIMULATION AND EXPERIMENTAL RESULTS

In Chapter four, an N-Net structure to perform as a swing up controller was designed. In the same Chapter a combined PD swing up controller was also designed and implemented on the laboratory setup. The neural controller designed in Chapter four was implemented as a controller on the real Inverted Pendulum setup in Chapter five. In the design of the N-Net structure, the method of Levenberg-Marquardt Optimisation algorithm was used to train the network structure. The method demonstrated that it is an effective training algorithm and was used to train an N-Net. The N-Net's interpolation behaviour was tested on the data obtained from the DDP algorithm and found to be satisfactory. In this chapter, it will be investigated how well the N-Net, using its interpolation capability, performs as a controller, first in simulations and then on the real setup. Experimental results of the combined PD swing up controller are also presented.

6.1 Simulation results

After training, an N-Net structure returns the acquired knowledge in form of: weight vector matrices, the network topology, and the activation functions for respective units in the structure. This knowledge can therefore, be retrieved at any time, by presenting the N-Net with the expected inputs (see Figure 6.1). For simulation purposes, the Inverted Pendulum discrete model developed in Section 4.4.2 is used as the process, as opposed to the real laboratory setup.

6.1.1 Performance of the N-Net controller

Given the initial state vector comprising four states, $x(k) = [0 \quad 0 \quad \pi \quad 0]^T$, $k = 0$, the N-Net structure is used to calculate the value of the next control signal, $u(k)$. This value of $u(k)$, is then used in the discrete non-linear Inverted Pendulum model to calculate the next system states. The system states obtained are in turn fed into the N-Net structure to calculate the next control signal (see Figure 6.1). The process repeats until the time $k = N$, is reached. Therefore, by iterating a number of times, the N-Net structure will have calculated a control trajectory. The final state of the Inverted Pendulum will also be obtained. By so doing, four state trajectories are also obtained. *The computed trajectories for the control and the four states are shown in Figure 6.2. The time* step size h = 0.01s which was used in dynamic programming is maintained.

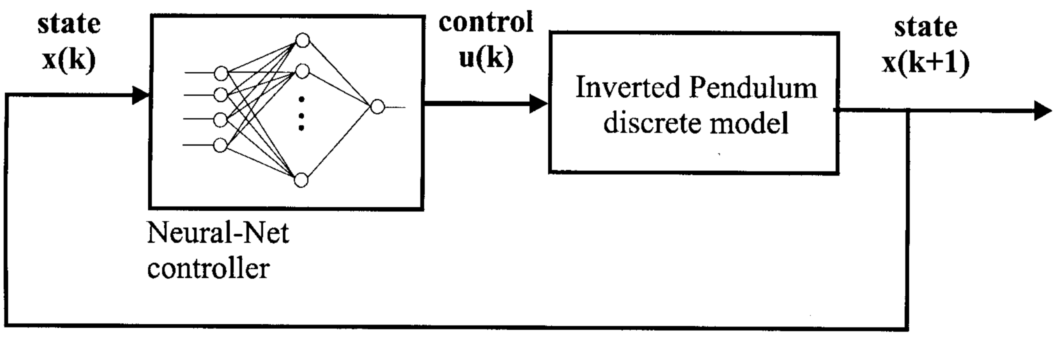


Figure 6.1 Simulating the N-Net controller

The result shows that the neural controller is able to steer the Inverted Pendulum from the downward stable resting position of $x(0) = [0 \quad 0 \quad \pi \quad 0]^T$, through a number of states, up to the final state of $x(N) = [0.03 \quad -1.39 \quad 3.05 \quad -9.44]^T$ in a total time of $T = 1.02$ s. Therefore, the neural controller has succeeded in generating energy in the Inverted Pendulum system. The target final state of the training data was $x(N) = [0.08 \quad -0.24 \quad 2.89 \quad -9.76]^T$. The greatest deviation is in $\dot{\varphi}$, otherwise the other states are very close to the target values. The difference in the final state of the N-Net and the target final state of the training data, especially $\dot{\varphi}$ itself is due to the fact that the MSE of the N-Net during training was not zero. In addition, the larger deviation in $\dot{\varphi}$ is inherent in the structure of the N-Net as explained in Section 4.5.3. But, as shown in the Figure 6.2, the target and N-Net output values are about the same. This means that, the N-Net controller is capable of steering the Inverted Pendulum in simulation, from the downward resting position, to a state which falls in (or near) the range of states controllable by the NL controller in the OS based controller.

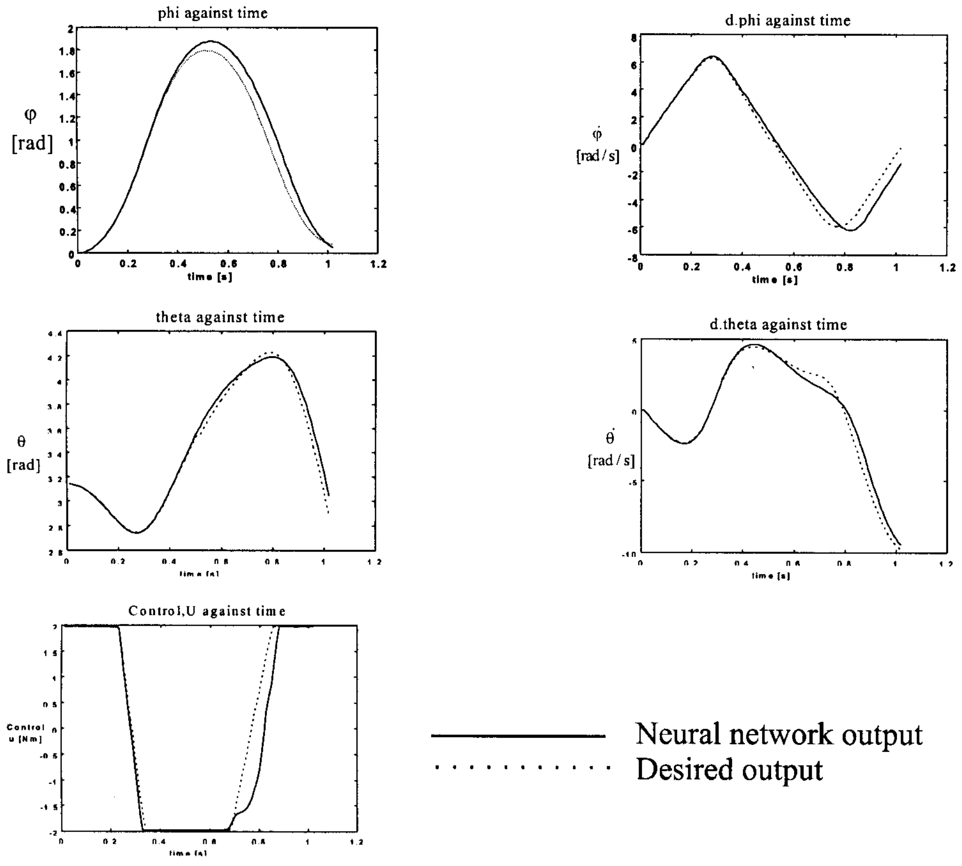


Figure 6.2 Simulation results of the N-Net structure applied as a non-linear swing up controller.

6.1.2 Sensitivity of the N-Net controller to variation in coulomb friction

Coulomb friction in the model of the Inverted Pendulum was varied to investigate the sensitivity of the swing-up controller to this parameter variation. The result is shown in Figure 6.3. It is observed that only large coulomb friction variations have a significant effect on the performance of the N-Net swing-up controller. However, coulomb friction of values around 0.2 Nm does not severely affect the operation of the swing up controller. In fact it enhances the controller's performance by reducing the final value of $\dot{\varphi}$ from $\dot{\varphi} = -1.39$ rad/s to $\dot{\varphi} = -0.83$ rad/s, a value which is much closer to the desired range of $-0.5 \le \dot{\varphi} \le 0.5$ rad/s.

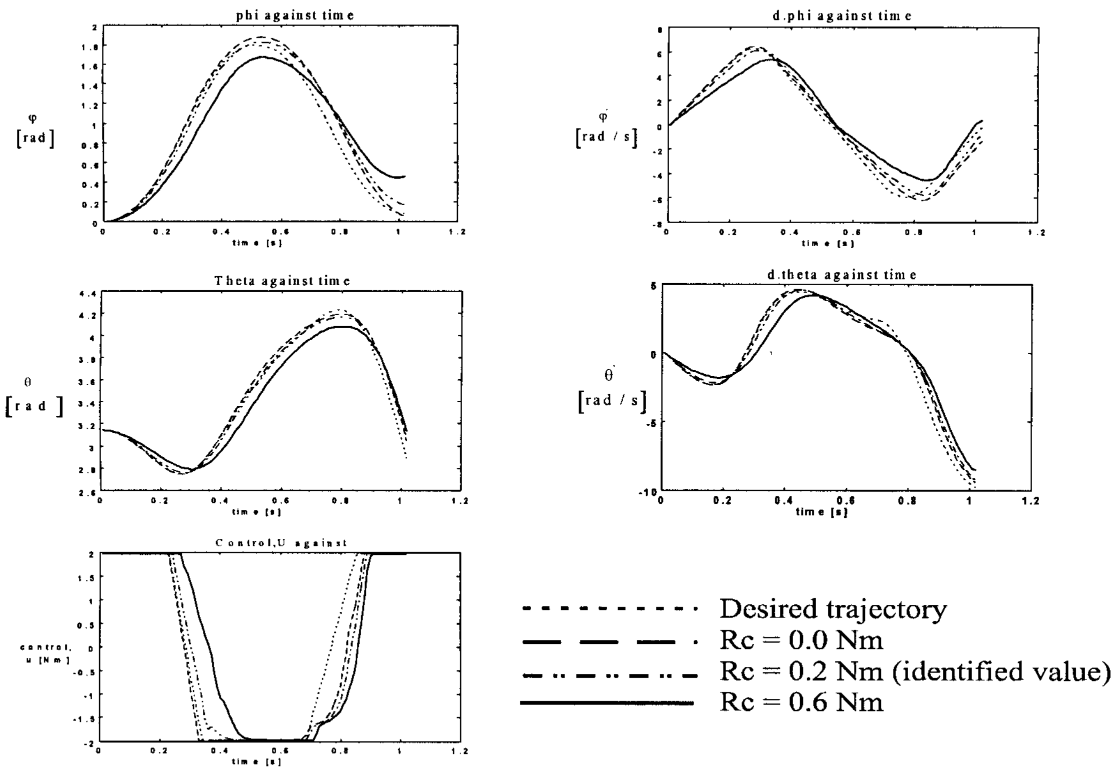


Figure 6.3 Investigating the effects of coulomb friction variation on the non-linear swing-up controller

The value Rc = 0.2 Nm is the identified value of coulomb friction on the Inverted Pendulum setup. Therefore, its existence on the setup is expected to have a positive effect on the performance of the swing-up controller. This result demonstrates the fact that the N-Net controller's performance remains impressive even in the wake of variations in values of coulomb friction which is bound to happen on the real setup.

6.1.3 Sensitivity to viscous friction (Rd) variation in the driving arm

The value of viscous friction in the driving arm (Rd), was varied in the model between 0 and 0.05 Nms. The results (see Figure 6.4) show that the performance of the controller does not deteriorate with variations in the values of viscous friction in this range. The extreme value of Rd = 0.05 Nms affects mostly the final states φ and $\dot{\varphi}$ but even then, not significantly. The conclusion is that the neural controller will still perform well when values of viscous friction drift away from the identified value of Rd = 0.002 Nms, but remain within the range 0 to 0.05 Nms. The extreme value of Rd = 0.05 Nms, is not even likely to be reached in practice.

6.1.4 Sensitivity to variations in viscous friction (Rs) variation in the balancing arm

To investigate the effect of variations in viscous friction in the balancing arm, the parameter was varied between Rs = 0 to Rs = 0.002 Nms. The identified value was 0.001 Nms. The results are shown in Figure 6.5. From the results, it is observed that small variations in this parameter have an effect on the final state achieved by the N-Net controller. The state $\dot{\varphi}$ is the most sensitive to the parameter variation. When compared with the sensitivity to viscous friction variations in the driving arm, the N-Net controller appears to be a lot more sensitive to variations in viscous friction in the balancing arm.

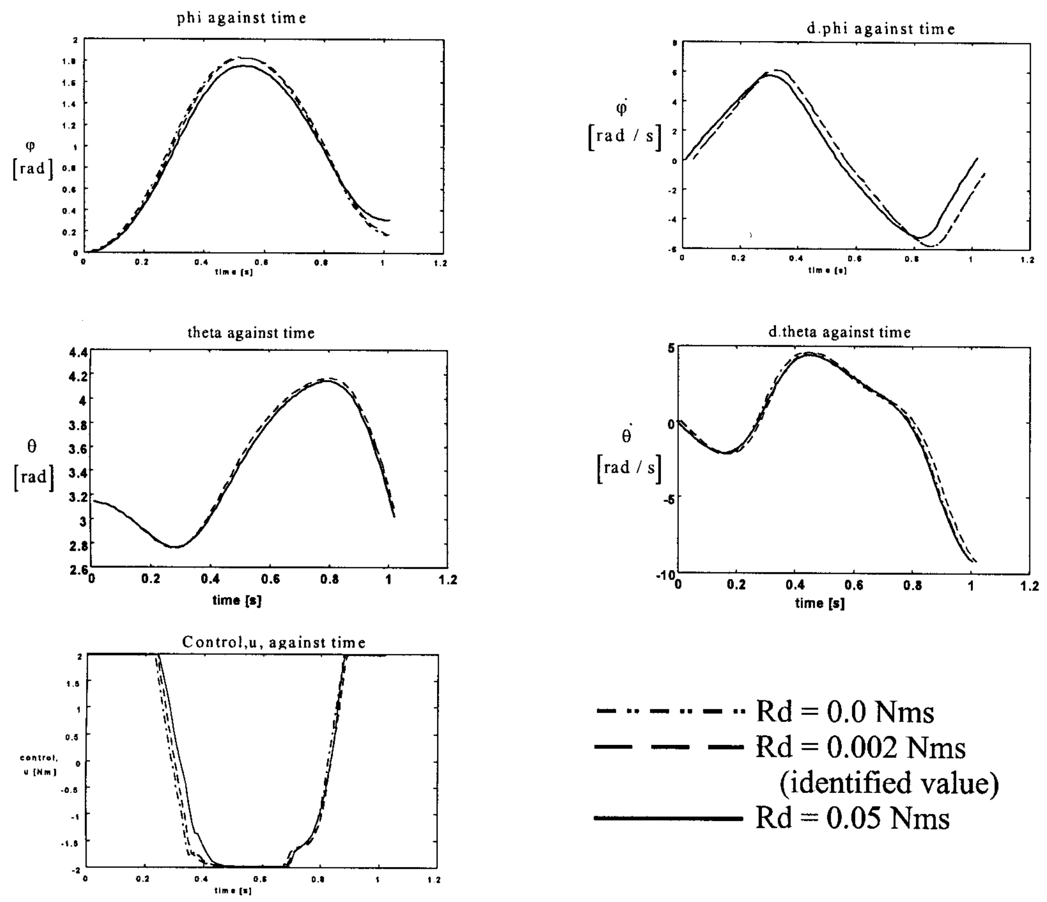


Figure 6.4 The effect of variations in viscous friction (Rd) of the driving arm

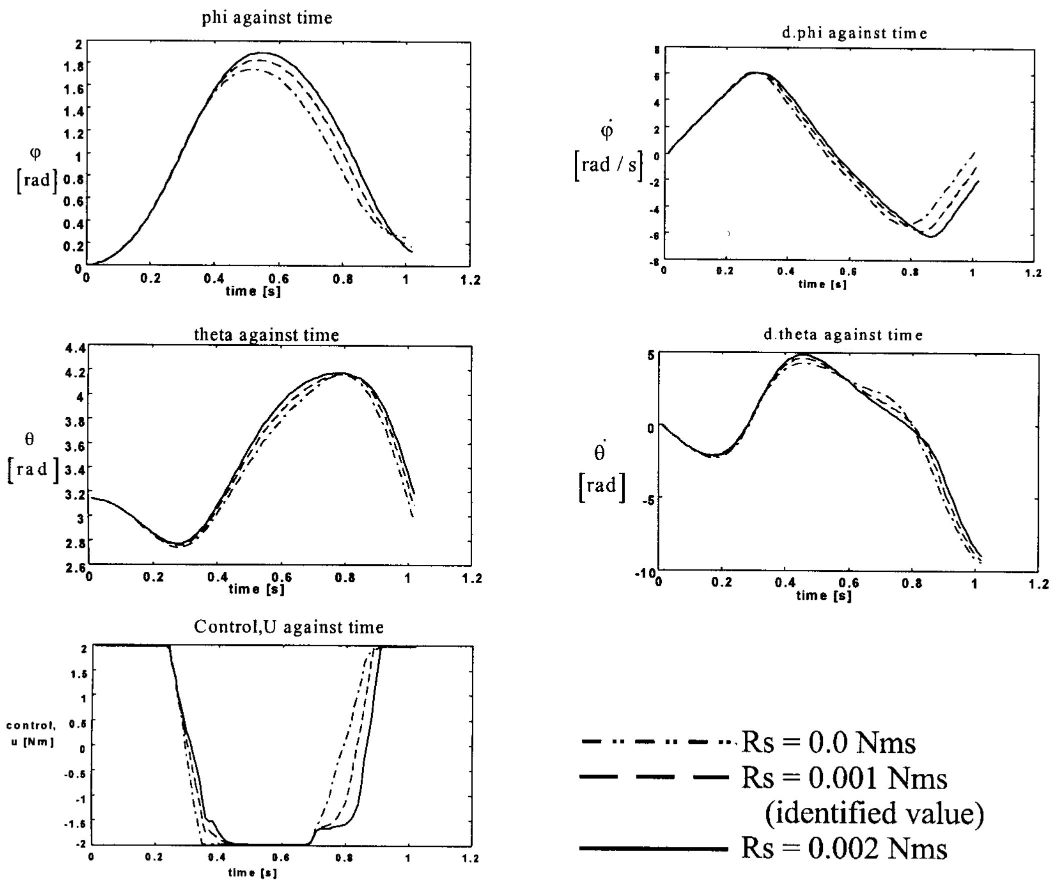
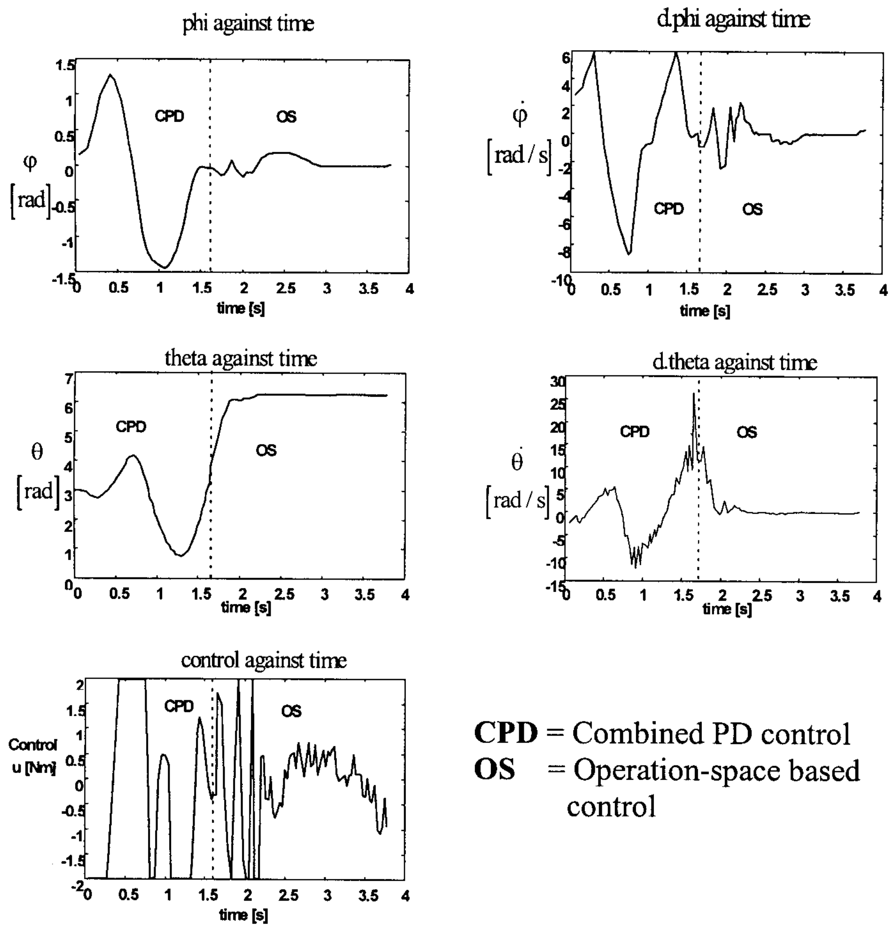


Figure 6.5 The effect of variations in viscous friction (Rs) of the balancing arm

6.2 *Experimental results*

6.2.1 Combined PD swing up controller

Results from the combined PD swing up controller which was developed in Section 4.3 are shown in Figure 6.6. The plots are generated from real data. The range of θ is defined on the interval $0 < \theta < 2\pi$. The combined PD swing up controller successfully steers the Inverted Pendulum from the downward resting position at $t \approx 0\text{s}$ to a state controllable by the OS based controller at $t \approx 1.7\text{s}$. The entry point is the NL controller in the form of the B-spline network. The results show that the absolute value of the maximum displacement of φ is less than 1.5 rad. The driving arm is first steered from the downward position to one side, then back past the downward position and to the other side, before being steered back to the downward position where control enters the OS based controller. The result demonstrates that compatibility of the combined PD swing up controller with the OS based controller has been achieved.



CPD = Combined PD control
OS = Operation-space based control

Figure 6.6 The combined PD swing up controller as part of the OS based controller

Further experiments to investigate the sensitivity of the PD swing up controller were performed. While in the downward resting position, the balancing arm is given a light swing just before the PD swing up controller is started. It was observed that depending on direction of the first swing relative to the direction of motion of the balancing arm, the controller will either fail or succeed to perform a swing up. If the first swing is in the same direction as the direction of motion of the balancing arm (e.g. both clockwise), then the controller is likely to fail. This is so because it has a damping effect on the system energy. This effect is inherent in the system model equations. Its implication is that the controller possesses little or no robustness. The failure point was observed to be during the back swing. The right initial state to perform a successful back swing will not be achieved. This behaviour is shown in Appendix F.

6.2.2 The N-Net controller

The N-Net controller was tested on the real Inverted Pendulum setup after the modification to the exponential function highlighted in Section 5.2. The controller generates correct control signals for the first half second or so, until the driving arm approaches the mechanical brake at $\varphi = 90$ deg. The displacement of the driving arm is not allowed to build up to the expected 103 deg (1.8 rad). Because of this disturbance, the N-Net is unable to reproduce the control trajectories as well as the state trajectories from this point on. This is shown in Figure 6.7.

The limit on displacement of the driving arm does not affect the operation of the OS based controller as well as the new combined PD swing up controller. Non of these controllers requires that the driving arm be displaced that much (see Figure 6.6). But the N-Net controller is inevitably hampered by the brake. In future, the controller design should take this constraint into account.

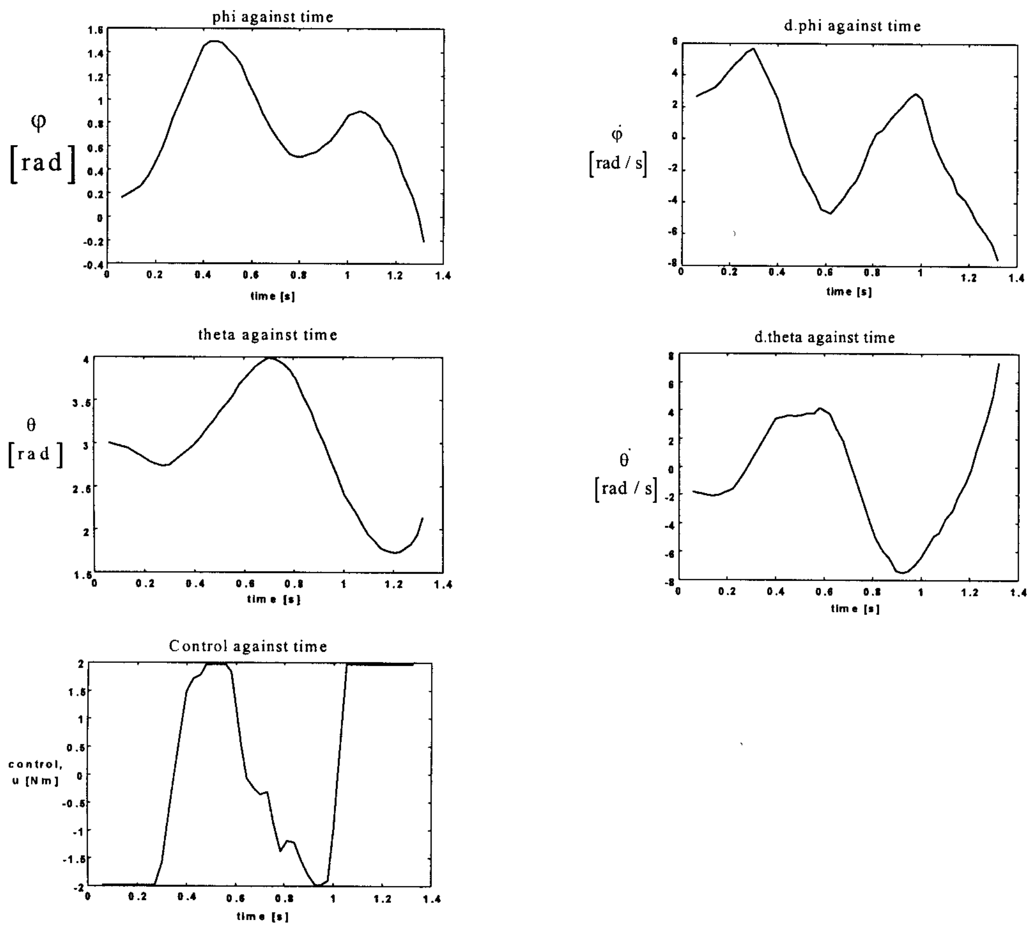


Figure 6.7 Performance of the N-Net controller on the real Inverted Pendulum setup

6.3 Conclusions

Results highlighted in this Chapter lead to the following conclusions:

- The designed N-Net controller performs very well as a swing up controller in simulations.
- The controller is not sensitive to almost all system parameter variations except to variations in viscous friction in the balancing arm.
- In practice, the N-Net controller's performance is hampered by physical hardware limitations. However, given the few seconds that the controller is allowed to operate before being

disturbed, it demonstrates that it is capable of reproducing the control and state trajectories in real time.

- The combined PD controller extends the Inverted Pendulum's controllable state space by including the downward resting position
- The successful transfer of control from the combined PD controller to the OS based controller in practice demonstrates the compatibility of the two controllers. It also shows that the N-Net controller should be compatible as shown in the simulations.

7.0 CONCLUSIONS AND RECOMMENDATIONS

7.1 Conclusions

A new operation-space based controller has been developed on the Inverted Pendulum setup. The controller has been developed by adding a fourth controller to the existing three controllers. The fourth controller of the OS based controller has been designed in two types. The first one is a combination of three PD controllers while the second one is an N-Net controller.

The combined PD controller successfully steers the Inverted Pendulum from the downward resting position and enters the existing OS based controller on the real setup. This is made possible due to saturation in the three PD controllers. The designed N-Net controller performs very well in simulations, and successfully steers the Inverted Pendulum from the downward resting position into a state controllable by the old OS based controller. The controllers is not sensitive to system parameter variations. The N-Net controller has also been implemented on the practical Inverted Pendulum setup. However, a mechanical brake which limits the movement of the driving arm, prevents the N-Net controller from fulfilling its control action. Also, the good interpolation behaviour of the N-Net observed in simulations is, on the real setup affected by approximations made in computations to avoid overflow errors.

An N-Net can be implemented as a non-linear controller if control knowledge is available. This can be done by training the N-Net with the control data using supervised training methods. The N-Net after being trained stores the acquired control knowledge in form of its topology, weight vector matrices and the kinds of activation functions for the various units in the ANN structure. This acquired knowledge may be retrieved and utilised at any time by a set of computations.

One of the limiting factors in the application of N-Nets as controllers is the strength of effectiveness of the training algorithm used. If the training algorithm is not good enough, small MSEs will not be achieved, resulting in N-Nets will poor interpolation behaviours.

The greatest limiting factor in the application of the N-Net to control the Inverted Pendulum (and in other control applications in general) lies in the generation of control knowledge. If the control knowledge is to be obtained by the Differential Dynamic Programming numerical method, then the limiting factor becomes the choice of the cost function(s). Since there are no guide lines available, the choice of the cost function consumes most of the design time. Some control tasks cannot be performed with one cost function. For instance, swinging the Inverted Pendulum from the downward resting position requires two cost functions. The problem of finding the cost functions then becomes even more time consuming. Considering the difficulties faced in arriving at appropriate cost functions for the swing up controller of the Inverted Pendulum, it may be impossible to find proper cost functions in other complex applications. In such cases, N-Nets may not be applied.

In control applications where control knowledge can be made readily available, the method of applying the N-Net can be very rewarding. Depending on the training algorithm used, the N-Net controller can perform very well against system parameter variations. This quality originates from the interpolation capability of the N-Net.

In practical implementation of N-Nets as controllers, increased forward calculation speed can be achieved by training N-Nets with a smaller number of hidden units. The desired small MSEs may be compensated for by effective training algorithms such as the Levenberg-Marquardt Optimisation method [13][15]. The application of N-Nets as controllers may also be rendered non feasible by the choice of activation functions, whose evaluation may not be handled by the available hardware and software resources.

7.2 Recommendations

Since the designed N-Net controller could not operate on the real setup due to the physical limitation on the displacement of the driving arm, the mechanical brake should be moved to a new position to allow for more movement of the driving arm. This operation must include corresponding modifications to the software implemented safety brakes in the Occam and Borland C++ source codes. Some practical considerations to avoid approximations of the exponential function during evaluation, should be sought. The approximation, as explained earlier, does affect negatively the interpolation behaviour of the N-Net. In the event that the N-Net controller still fails to reproduce the learned trajectories even after moving the mechanical brake, a few more points around the initial states of the second DDP trajectory should be included in the design. In addition, the signals obtained from the combined PD controller could be utilised to improve the performance of the N-Net controller. The N-Net controller could then be applied as a learning feed-forward controller.

APPENDICES

Appendix A Estimates of model parameters

The Inverted Pendulum used in this research is a copy of the original setup. Estimates of the model parameters for the original setup are listed below [3]. There is a possibility of slight variations in model parameters on the second setup. However, the second setup was also assumed to have the same model parameters, because controllers designed for the first setup performed equally well on the second setup. The parameters for which these values are given are defined in Figure 1.1.

mass moment of inertia of driving arm with respect to (2):	J_d=0.07 kgm^2
mass moment of inertia of balancing arm with respect to (3):	J_s=0.0023 kgm^2
length of driving arm	l_b=0.20 m
axis position of the mass centre times mass of driving arm:	$l_d m_d$=0.025
length of balancing arm:	l_s=0.165 m
mass of balancing arm:	m_s=0.15 kg
coefficient of viscous damping on the shaft (2):	R_d=0.002 Nms
coefficient of viscous damping on the shaft (3):	R_s=0.001 Nms
stiction:	R_c=0.20 Nms

Appendix B Iterative equations for the DDP algorithm

The objective of applying dynamic programming in control, is to choose an optimal value of the control u(k) which will minimise the cost function, *J*, at each stage of the multi-stage decision process [22]. This is done by determining the amount $\Delta u(k)$, by which to adjust the previously calculated control value, using (B2). The new value of the control is determined by updating the corresponding control value in the last adjacent control trajectory, by applying gradient descent, as given in (B1). Thus:

$$u_{i+1}(k) = u_i(k) + \alpha \Delta u(k) \qquad 0 < \alpha \le 1 \tag{B1}$$

Where

$$\Delta u(k) = -\frac{\delta g(x_i(k), u_i(k))}{\delta u_i(k)} - \left[\frac{\delta V(x_i(k+1), u_i(k+1))}{\delta x_i(k+1)}\right]^T \left[\frac{\delta x_i(k+1)}{\delta u_i(k)}\right] \tag{B2}$$

and α is the learning coefficient which is determined by the ratio between the actual change in the cost $V(x_{i+1}(0), u_{i+1}(0))$ and the estimated change in the cost $a(x_i(0), u_i(0), u_{i+1}(0))$ given by (B6). The terms in (B2) are evaluated recursively from the following set of equations:

$$\frac{\delta v(x_i(k), u_i(k))}{\delta x_i(k)} = \frac{\delta g(x_i(k), u_i(k))}{\delta x_i(k)} + \left[\frac{\delta V(x_i(k+1), u_i(k+1))}{\delta x_i(k+1)}\right]^T \left[\frac{\delta x_i(k+1)}{\delta x_i(k)}\right] \tag{B3}$$

$$V(x_i(k), u_i(k)) = g(x_i(k), u_i(k)) + V(x_i(k+1), u_i(k+1)) \quad \text{-(the cost function)} \tag{B4}$$

$$x(k+1) = f(x(k), u(k)) \qquad x(0)=x_0 \qquad x(N)=x_N \tag{B5}$$

$$a\big(x_i(k),u_i(k),u_{i+1}(k)\big) = a\big(x_i(k+1),u_i(k+1),u_{i+1}(k+1)\big)$$
$$-\alpha\left[\frac{\delta H\big(x_i(k),u_i(k),V\big(x_i(k+1),u_i(k+1)\big)\big)}{\delta u_i(k)}\right]^T\left[\frac{\delta H\big(x_i(k),u_i(k),V\big(x_i(k+1),u_i(k+1)\big)\big)}{\delta u_i(k)}\right] \quad \text{(B6)}$$

and the boundary conditions:

$$a\big(x_i(N)\big) = 0 \qquad \frac{\delta V(x_i(N)}{\delta x_i(N)} = \frac{\delta\Phi\big(x_i(N)\big)}{\delta x_i(N)} \quad \text{(B7)}$$

H() is the Hamiltonian function defined by:

$$H\big(x_i(k),u_i(k),V\big(x_i(k+1),u_i(k+1)\big)\big) = g\big(x_i(k),u_i(k)\big) + \left[\frac{\delta V\big(x_i(k+1),u_i(k+1)\big)}{\delta x_i(k+1)}\right]^T x_i(k+1)$$

Appendix C Pseudo code for the DDP algorithm

The general structure of the DDP algorithm is given below [4].

```
BEGIN
   initialise  Jstop;
   $x_0(0) = x_0$;
   $\underline{u}_0 = [u_0(0)\ u_0(1)\ \cdots\ u_0(N-2)\ u_0(N-1)]$;
   FOR  k=0 to N-1  DO
      calculate  $x_i(k+1)$;                                  (B5)
   FOR  k=N-1 to 0  DO
      calculate  $V(x_i(k),u_i(k))$;                          (B4)
   WHILE  $(V(x_i(0),u_i(0)) > \text{Jstop})$  DO
      BEGIN
         $\alpha = 1$;
         $a(x_i(N)) = 0$;                                     (B7)
         calculate  $\frac{\delta V(x_i(N),u_i(N))}{\delta x_i(N)}$;          (B7)
         FOR k=N-1  to 0  DO
            calculate $a(x_i(k),u_i(k),u_{i+1}(k))$;          (B6)
            calculate  $\frac{\delta V(x_i(k),u_i(k))}{\delta x_i(k)}$;       (B3)
         FOR  k=0 to N-1  DO
            calculate  $\Delta u(k)$;                         (B2)
            calculate  $u_{i+1}(k)$;                          (B1)
            calculate  $x_{i+1}(k)$;            /*  $x_{i+1}(k) = x_i + \Delta x$  */
            calculate  $x_{i+1}(k+1)$;                        (B5)
         FOR  k=N-1 to 0  DO
            calculate  $V(x_{i+1}(k),u_{i+1}(k))$;            (B4)
         $\Delta V(x_{i+1}(0),u_{i+1}(0)) = V(x_{i+1}(0),u_{i+1}(0)) - V(x_i(0),u_i(0))$;
         WHILE  $\left(\left[\frac{|\Delta V(x_{i+1}(0),u_{i+1}(0))|}{a(x_i(0),u_i(0),u_{i+1}(0))}\right] < c\right)$;          $0 < c \le 1$
            BEGIN
               $\alpha = \frac{\alpha}{2}$;          /* adaptation of the learning coefficient */
               FOR  k=0 to N-1  DO
                  calculate  $\Delta u(k)$;                   (B2)
                  calculate  $u_{i+1}(k)$;                    (B1)
```

```
                calculate   $x_{i+1}(k)$;                     /* $x_{i+1}(k) = x_i + \Delta x$ */
                calculate   $x_{i+1}(k+1)$;                          (B5)
            FOR  k=N-1 to 0  DO
               calculate   $V(x_i(k), u_i(k))$;                      (B4)
          $\Delta V(x_{i+1}(0), u_{i+1}(0)) = V(x_{i+1}(0), u_{i+1}(0)) - V(x_i(0), u_i(0))$;
         END
     i=i-1;
   END
END
```

Appendix D Derivation of the Inverted Pendulum discrete models

The iterative equations for the DDP algorithm require that the process be modelled by a set of difference equations as: $x(k+1) = f(x(k), u(k))$ $x(0)=x_0$ $x(N)=x_N$. Given the continuos time model (1.1), discrete models can be generated by numerical integration methods. Considering that dynamic programming is a complex computation method a complex discrete model would make the computation even more complex. So, it is desired to have as simple the discrete model as possible. On the other hand, the accuracy of the discrete model is vital, given that the design process is model based. The corrected van Luenen model (D1) was discretised by the first order Euler and the second order Runge-Kutta methods. The derivations are shown below.

$$T = M(\varphi,\theta)\ddot{v} + C(\varphi,\theta,\dot{\varphi},\dot{\theta}) + G(\varphi,\theta) + R(\dot{\varphi},\dot{\theta}) \qquad \text{(D1)}$$

Where

$$M = \begin{bmatrix} Jd + msl_b^2 - mslslb\cos(\varphi-\theta) & Js + msl_s^2 - mslslb\cos(\varphi-\theta) \\ -mslslb\cos(\varphi-\theta) & Js + msl_s^2 \end{bmatrix}$$

$$C = \begin{bmatrix} -m_s l_s l_b \sin(\varphi-\theta)\dot{\theta}^2 + m_s l_s l_b \sin(\varphi-\theta)\dot{\varphi}^2 \\ m_s l_s l_b \sin(\varphi-\theta)\dot{\varphi}^2 \end{bmatrix}$$

$$G = \begin{bmatrix} -g(l_d m_d + l_b m_s)\sin\varphi - g l_s m_s \sin\theta \\ -g l_s m_s \sin\theta \end{bmatrix}$$

$$R = \begin{bmatrix} (R_d + R_s)\dot{\theta} - R_c sign\dot{\varphi} \\ -R_s\dot{\varphi} + R_s\dot{\theta} \end{bmatrix}$$

$$\ddot{v} = [\ddot{\varphi} \;\; \ddot{\theta}]^T$$

and

$$T = \begin{bmatrix} T1 & T2 \end{bmatrix}^T$$

D.1 Model discretisation by Euler numerical method

Before discretizing the model in (D1) it was necessary to re-organise the model equation in terms of the vector $\ddot{v} = [\ddot{\varphi}, \ddot{\theta}]^T$. If we let

$$M = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}, \quad C = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \quad G = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} \quad \text{and} \quad R = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \tag{D2}$$

then,

$$\ddot{v} = \begin{bmatrix} \ddot{\varphi} \\ \ddot{\theta} \end{bmatrix} = \frac{1}{|M|} \begin{bmatrix} m_{22} & -m_{12} \\ -m_{21} & m_{11} \end{bmatrix} \left[\begin{bmatrix} u \\ 0 \end{bmatrix} - \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} - \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} - \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \right] \tag{D3}$$

where |M| is the determinant of matrix M defined by : $|M| = m_{11} * m_{22} - m_{12} * m_{21}$

If we let $\begin{bmatrix} u \\ 0 \end{bmatrix} - \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} - \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} - \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} ucgr1 \\ ocgr2 \end{bmatrix}$ then

$$\ddot{\varphi} = \frac{1}{|M|} \left[m_{22} * ucgr1 - m_{12} * ocgr2 \right] \qquad \text{(D4}$$

$$\ddot{\theta} = \frac{1}{|M|} \left[-m_{21} * ucgr1 - m_{11} * ocgr2 \right] \tag{D5}$$

Euler integration is obtained from Euler differentiation given by [24][26][27]:

$$\dot{x}_k \approx \frac{x_{k+1} - x_k}{h} \tag{D6}$$

$$\Rightarrow x_{k+1} = h * \dot{x}_k + x_k, \qquad \text{where } h = \text{the time step size or sampling interval} \tag{D7}$$

Let us define the 4 states as x_1, x_2, x_3 and x_4. Then we have:

$$x_1 = \varphi$$

$$\dot{x}_1 = \dot{\varphi} = x_2$$

$$\dot{x}_2 = \ddot{\varphi}$$

$$x_3 = \theta$$

$$\dot{x}_3 = \dot{\theta} = x_4$$

$$\dot{x}_4 = \ddot{\theta}$$

which yields 4 first order differential equations as follows:

$$\dot{x}_1 = x_2 \tag{D8}$$

$$\dot{x}_2 = \ddot{\varphi} = \frac{1}{|M|}\left[m_{22} * ucgr1 - m_{12} * ocgr2\right] \tag{D9}$$

$$\dot{x}_3 = x_4 \tag{D10}$$

$$\dot{x}_4 = \ddot{\theta} = \frac{1}{|M|}\left[- m_{21} * ucgr1 - m_{11} * ocgr2\right] \tag{D11}$$

Applying Euler integration (D7) to equations (D8) to (D11) yields 4 iterative equations for calculating the next state for each of the 4 states in the form:

$$x_1(k+1) = h * x_2(k) + x_1(k) \qquad x_1(0) = x_{1_0} \tag{D12}$$

$$x_2(k+1) = h * \ddot{\varphi}(k) + x_2(k) \qquad x_2(0) = x_{2_0} \tag{D13}$$

$$x_3(k+1) = h * x_4(k) + x_3(k) \qquad x_3(0) = x_{3_0} \tag{D14}$$

$$x_4(k+1) = h * \ddot{\theta}(k) + x_4(k) \qquad x_4(0) = x_{4_0} \tag{D15}$$

Hence with an initial state vector $x_0 = [x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}]^T$ and a control input vector *u*, it is possible to determine the state at any time *k* using equations (D12) to (D15).

D.2 Model discretisation by Runge-Kutta numerical method

The method is similar to Euler. The difference is that in the Runge-Kutta method, the next state $\hat{x}_{k+1}$ is first estimated the by Euler method. Then a derivative at this value of the state is estimated. Next, the new state x_{k+1} is determined from the average of the two derivatives [26]. That is:

$$k_1 = f(x_k, u_k) \tag{D16}$$

$$k_2 = f(x_k + k_1 h, u_{k+1}) \tag{D17}$$

$$x_{k+1} = x_k + \frac{h}{2}(k_1 + k_2) \tag{D18}$$

Transforming (D1) into the form (D12) to (D15) requires more steps than in the Euler method. It was done as follows:

$$k_1 x_1 = \dot{x}_1(k) = x_2(k) \tag{D19}$$

$$k_1 x_2 = \dot{x}_2(k) = \ddot{\varphi}_k = f(x_1(k), x_2(k), x_3(k), x_4(k), u(k)) \tag{D20}$$

$$k_1 x_3 = \dot{x}_3(k) = x_4(k) \tag{D21}$$

$$k_1 x_4 = \dot{x}_4(k) = \ddot{\theta}_k = f(x_1(k), x_2(k), x_3(k), x_4(k), u(k)) \tag{D22}$$

The $\boldsymbol{k_2}$ values are determined as follows:

$$k_2 x_1 = x_2(k) + k_1 x_2 * h \tag{D23}$$

$$\begin{aligned} k_2 x_2 &= \ddot{\varphi}_{k+1} \\ &= f((x_1(k) + k_1 x_1 * h), (x_2(k) + k_1 x_2 * h), (x_3(k) + k_1 x_3 * h), (x_4(k) + k_1 x_4 * h), u(k+1)) \end{aligned} \tag{D24}$$

$$k_2 x_3 = x_4(k) + k_1 x_4 * h \tag{D25}$$

$$\begin{aligned} k_2x_4 &= \ddot{\theta}_{k+1} \\ &= f\left((x_1(k)+k_1x_1*h),(x_2(k)+k_1x_2*h),(x_3(k)+k_1x_3*h),(x_4(k)+k_1x_4*h),u(k+1)\right) \end{aligned} \tag{D26}$$

Finally, the next states are determined from:

$$x_1(k+1) = x_1(k) + \frac{h}{2}(k_1x_1 + k_2x_1) \tag{D27}$$

$$x_2(k+1) = x_2(k) + \frac{h}{2}(k_1x_2 + k_2x_2) \tag{D28}$$

$$x_3(k+1) = x_3(k) + \frac{h}{2}(k_1x_3 + k_2x_3) \tag{D29}$$

$$x_4(k+1) = x_4(k) + \frac{h}{2}(k_1x_4 + k_2x_4) \tag{D30}$$

Equations (D19) to (D30) are the iterative set of equations with which any next state may be determined for a given initial state vector $x(0)$, and the control vector u , at any time k.

Appendix E Convergence by bisection method

The bisection method , also known as the binary search method was used to find the minimum number of time steps N in which the DDP cost function reaches the minimum. The method was implemented as part of the DDP algorithm. A handshake signal **ok** was defined. If the DDP algorithm reaches the minimum cost (Jstop), the value of ok=1 is returned. In this case the number of time steps N must be reduced. Otherwise, if the returned value of ok=0, then the DDP algorithm failed to reach the minimum cost in N time steps. The number of time steps must therefore be increased. The value by which to increase or decrease the number of time steps is calculated from the defined values of maximum and minimum time steps, N_max and N_min respectively [4]. This is illustrated in below:

```
BEGIN
   initialise min_N;
   initialise max_N;
   number_of_loops=(log(max_N-min_N)/log2);
   FOR  I=0 to number_of_loops  DO
      BEGIN
         N= (min_N+(max_N-min_N)/2);
         ok=0;
         ok=calctraject(..,N,..);  /* call DDP algorithm */
         IF (ok=1) then
            max_N=N;              /* minimum cost reached */
         ELSE
            min_N=N;
         END
      END
   NEXT I
END
```

Appendix F: Failure of the Combined PD swing up controller.

The combined PD swing up controller is able to swing the Inverted Pendulum, but is not robust. The controller fails for slight deviations in the initial states.

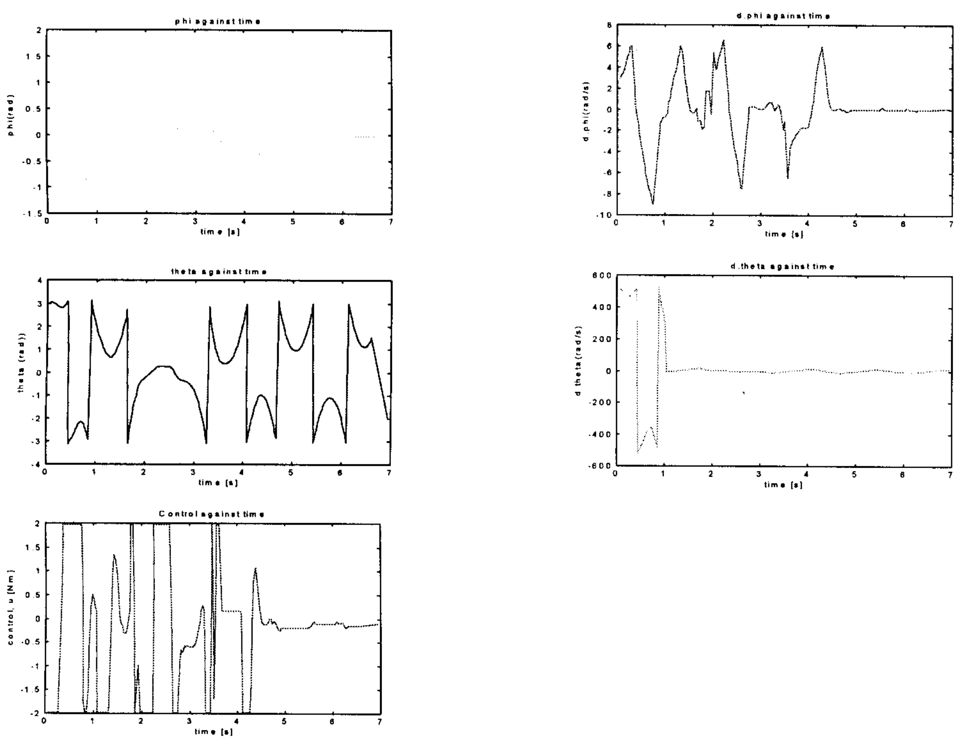


Figure F.1 Failure of the combined PD swing up controller (real data)

In Figure F1, it is observed that the plot of φ is more or less the same as in Figure 6.6 for the duration of the PD swing up controller. The implication is that the driving arm swings to the left and right (in this case to the right then left) just like in the first case. However, examination of the plot of θ reveals that something went wrong during the second swing. The value of θ (at t =1.8s) is less, implying that the balancing arm did not deflect far enough to have the right velocity at the point of entry into the OS-based controller. Nevertheless, small enough swings in the balancing arm do not affect the performance f the PD swing up controller. Note that θ and its derivative $\dot{\theta}$ are plotted for the interval $-\pi < \theta < \pi$.

REFERENCES

[1] Spong M. W., " *The Swing up Control Problem For The Acrobot*" IEEE Control Systems Magazine, Volume 15 No. 1, pp49-55, 1995.

[2] Jager de P.J. " *User interface and software for the balancing stick system*" , Internal report no. 92R192, Control Laboratory, Department of Electrical Engineering, University of Twente, Enschede, Netherlands, 1992.

[3] Oosterveen H. , " *Design and Implementation of a State Feedback and Neural Controller for the Inverted Pendulum*", MSc thesis, report no. 89R140, Control Laboratory, Department of Electrical Engineering, University of Twente, Enschede, The Netherlands.

[4] Jongma J.A. " *Non linear control of an Inverted Pendulum using N-Net concepts*", MSc thesis, report no. 93R120, Control Laboratory, Department of Electrical Engineering, University of Twente, Enschede, Netherlands.

[5] Luenen van W.T.C. , " *Neural Networks for control - on knowledge representation and learning*" , PhD thesis, University of Twente, Enschede, The Netherlands, 1993.

[6] Krijgsman A.J. "*Model-based and intelligent control*". PhD thesis, Delft University of Technology, Delft, 1994.

[7] Boullart L. , Krijgsman A. And Vingerhoeds R.A. , "*Application of Artificial Intelligence in Process Control*". Pergamon Press Ltd, Oxford, England, 1992.

[8] Hunt K.J. , Sbarbaro D. , Zbikowski R. , and Gawthrop P. J. , "*Neural Networks for Control Systems - A survey*". International Federation of Automatic Control. Vol. 28, No 6, pp 1083-1112, 1992.

[9] White D. A. , and Sofge D. A. , *"Handbook of Intelligent Control - Neural, Fuzzy, and adaptive approaches"*. Multiscience Press, Inc. , New York, USA, 1992. ✓

[10] Kröse B.J.A. and Smagt der van P. "*An introduction to N-Nets*". University of Amsterdam, Netherlands, 1995. ✓

[11] Miller III W.T. , Sutton S.R. , and Werbos, P. J. , *"Neural Networks for control"*. The MIT Press, London, England, 1995. ✓

[12] Zornetzer S.F. , Davis J. L. , Lau C. , and McKenna T. , *"An Introduction to Neural and Electronic Networks"*. 2nd Edition , Academic Press, Inc. , London, UK, 1995. ✓

[13] Fletcher R. , *"Practical Methods of Optimization"*, John Wiley and Sons, Inc. , 1987. ✓

14] Cybenko G. , "*Approximation by superposition of a sigmoid function*". Mathematics of control, signals and systems, 2:303-314, 1989. ✓

15] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, *"Numerical Recipes in C: The Art of Scientific Computing"* , 2nd Edition, Cambridge Univ. Press, N.Y., 1992. ✓

16] Brown M. And Harris C. J. , "*A non-linear adaptive controller: A comparison between fuzzy logic control and neuro-control*". ✓

17] Wang X. L. , *"Adaptive Fuzzy Systems and Control - design and stability analysis"*. PTR Prentice Hall, Inc. , New Jersey, 1994. ✓

8] Jacobson D.H. and Mayne D.Q. , "*Differential Dynamic Programming*". American Elsevier Publishing Company, Inc, New York, 1970. ✓

9] Bellman R. , "*Dynamic Programming*". Princeton University Press, Princeton, N.J. , 1957. ✓

0] Bellman R. , "Applied *Dynamic Programming*". Princeton University Press, Princeton, N.J. , 1962.

[21] Bellman R. and Angel E. , "*Dynamic Programming and Partial Differential Equations*". Mathematics in Science and Engineering, Volume 88. Academic Press, New York, USA , 1972. ✓

[22] Jacobs O.L.R. , *"An Introduction to Dynamic Programming - The Theory of Multistage Decision Processes"*. Chapman and Hall Ltd. , The Netherlands, 1967. ✓

[23] Åström J. K. And Wittenmark B. , *"Computer Controlled Systems - Theory and Design"*. 3rd Edition, Prentice Hall Inc. , New Jersey, 1997. ✓

[24] Dorf C. R. and Bishop H. R. , *"Modern Control Systems"*. 7 th Edition, Addison-Wesley Publishing Company, Inc. , USA, 1995. ✓

[25] Beightler S. C. , Phillips T. D. , and Wilde J. D. , *"Foundations of Optimisation"*. 2nd Edition, Prentice Hall, Inc. , New Jersey, 1979. ✓

[26] Kreyszig E. , *"Advanced Engineering Mathematics"*. John Wiley and Sons, Inc. , New York, USA, 3rd Edition, 1972. ✓

[27] Norman B. L. , *"Discrete Mathematics"*. Oxford University Press, New York, Revised Edition, 1989. ✓