PREDICTIVE CONTROL OF MULTIBODY SYSTEMS FOR THE SIMULATION OF MANEUVERING ROTORCRAFT

A Thesis
Presented to
The Academic Faculty

by

Yalcin Faik Sumer

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Aerospace Engineering

School of Aerospace Engineering
Georgia Institute of Technology
April 2005

PREDICTIVE CONTROL OF MULTIBODY SYSTEMS FOR THE SIMULATION OF MANEUVERING ROTORCRAFT

Approved by:

Prof. Carlo L. Bottasso, Chair,
Daniel Guggenheim School of
Aerospace Engineering,
Georgia Institute of Technology

Prof. Olivier Bauchau,
Daniel Guggenheim School of
Aerospace Engineering,
Georgia Institute of Technology

Prof. Dewey H. Hodges,
Daniel Guggenheim School of
Aerospace Engineering,
Georgia Institute of Technology

Date Approved: 14 April 2005

To my family and real friends

ACKNOWLEDGEMENTS

First of all, I want to thank my advisor Dr. Bottasso for his effort of teaching and guiding me to finish this MS thesis.

I would like to thank also my parents supporting me, by being here with me during the preparation of this thesis.

Furthermore, special thanks to my best friend Ayse Gul Gungor for her great support especially during the hard times in this education period.

Special thanks for also the following people for their main help during my studies ;

Luca Riviello for his advices, ideas and help during the research period.

Alessandro Croce and Domenico Leonello for their help in guiding me using the comprehensive codes.

My officemates Chong-Seok Chang, Julide Topsakal and Nazmiye Acikgoz for their daily support.

My dance instructor TJames Scott for his help in reviewing the thesis for errors.

My friend Zehra Konya for her support in both of my thesis.

Also thanks to all my friends...

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

$\widetilde{(\bullet)}$	Denotes quantities of the multibody model
$(\bullet)^{\text{track}}$	Denotes quantities of the model predictive tracking problem
$(\bullet)^{\text{steer}}$	Denotes quantities of the steering problem
$(\bullet)^{\text{plan}}$	Denotes quantities of the planning problem
$(\bullet)^{\text{adapt}}$	Denotes quantities of the model adaption problem
$(\bullet)^{*}$	Known assigned quantity
$(\bullet)_h$	Discretized quantity, as obtained by the use of a numerical method
t	Time
$\dot{(\bullet)} = \mathrm{d}(\bullet)/\mathrm{d}t$	Derivative with respect to time
T_0	Initial time
T	Final time
$\mathcal{T}_h$	Computational grid on the interval $[T_0, T]$
K	Element (time step) of $\mathcal{T}_h$
J	Cost function
$\widetilde{\boldsymbol{x}}$	Multibody states
$\widetilde{\boldsymbol{\lambda}}$	Multibody Lagrange multipliers
$\widetilde{\boldsymbol{u}}$	Multibody controls
$\widetilde{\boldsymbol{y}}$	Multibody outputs
$\boldsymbol{y}$	Reduced model states
$\boldsymbol{u}$	Reduced model controls
$\boldsymbol{p}$	Reduced model parameters

SUMMARY

Simulation of maneuvers with multibody models of rotorcraft vehicles is an important research area due to its complexity. During the maneuvering flight, some important design limitations are encountered such as maximum loads and maximum turning rates near the proximity of the flight envelope. This increases the demand on high fidelity models in order to define appropriate controls to steer the model close to the desired trajectory while staying inside the boundaries. The desired trajectory is dependent on the given mission or task. A framework based on the hierarchical decomposition of the problem is used for this study. The system should be capable of generating the track by itself based on the given criteria and also capable of piloting the model of the vehicle along this track. The generated track must be compatible with the dynamic characteristics of the vehicle. Defining the constraints for the maneuver is of crucial importance when the vehicle is operating close to its performance boundaries.

In order to make the problem computationally feasible, two models of the same vehicle are used where the reduced model captures the coarse level flight dynamics, while the fine scale comprehensive model represents the plant. The problem is defined by introducing planning layer and control layer strategies. The planning layer stands for solving the optimal control problem for a specific maneuver of a reduced vehicle model by satisfying the given constraints and optimizing the cost function. The control layer takes the resulting optimal trajectory as an optimal reference path, then tracks it by using a non-linear model predictive formulation and accordingly steers the multibody model by solving the time marching problem with the given initial conditions. Reduced models for the planning and tracking layers are adapted by using neural network approach online to optimize the predictive capabilities of planner and tracker.

Optimal neural network architecture is obtained to augment the reduced model in the

best way. The methodology of adaptive learning rate is experimented with different strategies. Some useful training modes and algorithms are proposed for these type of applications. It is observed that the neural network increased the predictive capabilities of the reduced model in a robust way.

The proposed framework is demonstrated on a maneuvering problem by studying an obstacle avoidance example with violent pull-up and pull-down.

Key words: Multibody dynamics; Maneuvers; Trajectory optimization; Optimal control; Model predictive control; Neural Networks; Trajectory tracking; Flight mechanics; Vehicle dynamics

CHAPTER I

INTRODUCTION AND MOTIVATION

This study is concerned with maneuvering multibody dynamics (MMBD) by using an adaptive model predictive controller. MMBD is based on generating and executing a plan for flying a virtual prototype of a vehicle between specific locations, achieving a given task [1]. For any given task we can have certain constraints, obstacles, specific flight conditions, boundaries of a finite performance envelope and also some special requirements for the operational phase. The planning part of the problem should ensure the satisfaction of those task requirements. Current approaches for the multibody dynamics are based on computing the motion of the vehicle by integrating the model equations in time, by taking as input initial conditions and the given controls. The significant point here is the determination of the time histories of control inputs. It is very complicated and computationally expensive to produce the control inputs and the tracking trajectory by solving the optimal control problem (boundary value problem) for the multibody model. However, MMBD is concerned with generating the control actions for a given task by blending the coarse and fine scale models. Typically we can define the problem in two sections as a) how to operate the model of the vehicle based on the given criteria b) steer the model accordingly [1].

Great progress has been made in recent years towards the comprehensive simulation of the rotorcraft based on the high fidelity aeroelastic mathematical models. Those models are based on the multibody finite element methods that provide the ability to model the most complex part of the vehicle, the rotor system, in a detailed way [6]. Rotorcraft multibody codes are coupled with time-accurate aerodynamic models ranging from dynamic inflow to free wake models all the way to first principles of computational fluid dynamics [1, 7]. Generally multibody-based analysis rely on complex, large, highly non-linear multi-field models. These high fidelity mathematical models of rotorcraft systems are currently dealing with the analysis of hover and forward flight regimes. Already some procedures are available

such as constant-in-time control inputs that trim the aircraft model either in wind tunnel or free flight modes [5]. On the other hand, only flight mechanics models (reduced model, coarse model) are being used to perform a typical simulation of maneuvering flight [3]. In those models, vehicles are mostly modelled as a rigid body and the rotor is described by using blade element theory with wake corrections. In this sense, same physical system can be defined by two different mathematical models such as aeroelastic (comprehensive, fine scale) model and flight mechanics model which has far fewer degrees of freedom. Although aeroelastic models are able to render fine scale details of the solution, reduced model is blind to these small scales. However, those models are still able to capture the coarser scales of the physical processes in a sense of flight mechanics characteristics.

There can be also problems such that the trajectory of the vehicle is given or easy to determine. In this case, simulation of the maneuver becomes only a tracking problem where the controller steers the vehicle along a pre-assigned trajectory. However, for rotorcraft based applications, it is very difficult or almost impossible to guess a priori reasonable tracking paths. This is due to the fact that, there are some factors such as maximum loads and limiting criteria that are encountered during the maneuvering flight in proximity of the flight envelope. In those cases, it will be very difficult to know whether a given path is trackable, whether it is optimal or whether it satisfies the constraints based on performance and operational requirements. There are some different rotary wing civil applications such as emergency maneuvers following the partial loss of power due to an engine failure during the take-off and obstacle avoidance problem with violent pull-up/ pull-down [1, 2]. These studies addressed those issues by removing the assumption of pre-assigned track. However, those studies are concentrated on the parameter optimization problem in a sense of adjusting the reduced model. In this study, neural network is applied on the reduced model as an adaptive controller to generate a more robust case. When compared with the previous studies we can expect better tracking performance for the planned robust application.

The motion planning phase is totally based on the optimal control theory where we define the vehicle performance index as a cost function which should be minimized subjected to the system dynamics and other constraints on the controls and the states. Optimal control

requires the solution of boundary value problems rather than the classical initial value problems solved by time-marching multibody codes. By the increase of the complexity of the model, the computational cost becomes a serious problem for boundary value applications. In this sense, the maneuver optimal control problem is solved at the flight mechanics level which is inexpensive. Then the controls computed as part of the solution are used for steering the fine aeroelastic model. In this way, the fine level solution becomes a classical forward dynamics integration with acceptable computational cost. In order to ensure the convergence of the trajectories flown by the two models to a common result, iteration methodology, using an adaptive controller, has been employed between coarse and fine level representations to adjust the flight mechanics model to behave as close as to the aeroelastic model.

Here is the detailed hierarchy of the layers that are used for the path planning and path tracking [1] :

- A *strategic layer* is concerned with the definition of the problem regarding the tasks that need to be studied and the final objectives of the process. (For example, a helicopter avoiding an obstacle should accomplish violent pull up and accordingly violent pull down in minimum time to recover the mission altitude and speed while satisfying the constraints on controls and trajectory). The output of this layer will be the definition of the maneuver as an optimal control problem in terms of cost function and constraints.

- A *tactical layer* is responsible for the navigation and guidance of the vehicle based on planning the best trajectory that satisfies the goal defined in the strategic layer. In this planning phase, a reduced vehicle model is used in solving the optimal control problem which is still able to capture the global dynamic characteristics (gross overall motion) of the plant, i.e. of the detailed multibody model. Computational costs are inexpensive since this coarse model has few degrees of freedom. The output of this layer is the tracking trajectory which will be an input for the next layer.

- A *reflexive layer* focuses on the control problem by implementing the necessary control

actions to the plant to track the trajectory created in the previous layer. Also the reduced model estimation is still used. This layer implements the non-linear model predictive control (NMPC) [11] to adjust the reduced model in a sense of matching the two models. Basically, the controller predicts the future behavior of the plant by using this reduced model and finds the control inputs that are necessary to steer the plant along the generated trajectory, solving the optimal control problem on a receding horizon. Since steering in open-loop is prone to instabilities, receding horizon methodology can be used as an application of model based predictive control for Multi-Model Steering Algorithm (MMSA) [2].

By using the receding horizon methodology as a control policy we can come up with stable behaviors of controls and states. Clearly, we are steering the aeroelastic model for a short period of time and then solving the optimization problem again on a time shifted horizon. At this time, adaption procedure is taking an important role on the non-linear reduced model in order to predict the dynamics of the plant closely and guarantee small tracking errors. These steps are iterated until we reach the end of the maneuver. The reflexive layer performance is efficiently increased with the adaption procedure.

The adaption strategy on the reduced model requires the on-line training of the neural network by using the information taken from the previous receding horizon window which will be the difference between the predicted and effectively realized vehicle behaviors.

For clarity, tactical planner and reflexive controller both use the reduced model to achieve low computational costs. Also as a a part of the reflexive layer, the multibody code takes the role of the plant and it is used for solving the initial value problem with known control actions.

Once the planned trajectory has been tracked by the reflexive controller, we have to correct the reduced model deficiency according to the achieved performance. Then we have to go back to the tactical layer and repeat the trajectory planning phase with the improved reduced model. Contributions between those two layers should be carried out by iteration in order to enforce the compatibility on the tracking trajectory of reduced model and plant until we get a convergence or no further model improvements. This adaptive planning and

tracking methodology uses MMSA.

As a first step, general information is given about the neural networks that can be useful for this study (chapter 2). In order to find the best approach for the identification of system errors, we worked on a simple dynamic problem (chapter 3). Then we described the adaptive tracking and steering of the multibody models with the given trajectories (chapter 4). As a next step, formulation of the maneuver optimal control problem is developed for the trajectory planning. Then different possible alternatives on implementing the adaptive planning and tracking procedures are introduced (chapter 4). In order to assess the performance, this methodology is applied to an obstacle avoidance problem involving violent pull-up / pull-down maneuvers and results of this robust methodology is compared with the previous studies (chapter 5). As a final step, optimal neural network architecture is proposed for these kind of problems (chapter 6).

CHAPTER II

NEURAL NETWORKS

Neural network is a highly complex, non-linear information processing system which is motivated by the human brain. It is designed to model the way in which the brain performs a particular task or function of interest. The vital importance lies under the learning capability of this structure. Once the required knowledge is supplied to the network from its environments through a learning process, it is capable of producing reasonable outputs for inputs not encountered during training. This property is called generalization capability of the neural network. By the use of this property, it is possible to solve complex (large scale) problems that are currently intractable.

Here are some other important capabilities and properties of Neural Networks.

Nonlinearity : An artificial neuron can be linear or non-linear. Interconnection of non-linear neurons create a non-linear structure. Nonlinearity is distributed through the network.

Input-output mapping : Throughout the learning period neural network is subjected to set of inputs and corresponding desired responses. Synaptic weights starting from initial values are modified to minimize the difference between the desired and actual responses. This procedure is repeated for many examples where network reaches steady state (no further improvements are possible) or until the desired error is achieved.

Adaptivity : Neural networks have a built-in capability to adapt their synaptic weights to changes in the surrounding environment. It can be retrained to deal with minor changes. In non-stationary environments (dynamic systems, ...), a neural network can be designed to change its synaptic weights in real time. Control applications coupled with the adaptive capability make it a useful tool especially in adaptive control. The more adaptive we make a system, the more robust its performance will likely be when operating in a stationary environment.

Fault tolerance : A neural network has the potential to be inherently fault tolerant or capable of robust computation. Thus, a neural network exhibits a graceful degradation in performance rather than catastrophic failure under adverse operating conditions. In order to be assured that the neural network is in fact fault tolerant, it may be necessary to take corrective measures in designing the algorithm to train the network.

2.1 Neural Network Architecture

Decision on the architecture of the neural network is one of the important issues that affects the overall performance of the application. Before looking at different architectures lets define the smallest element of this architecture which are neurons. Non-linear model of neuron can be seen from figure (1).

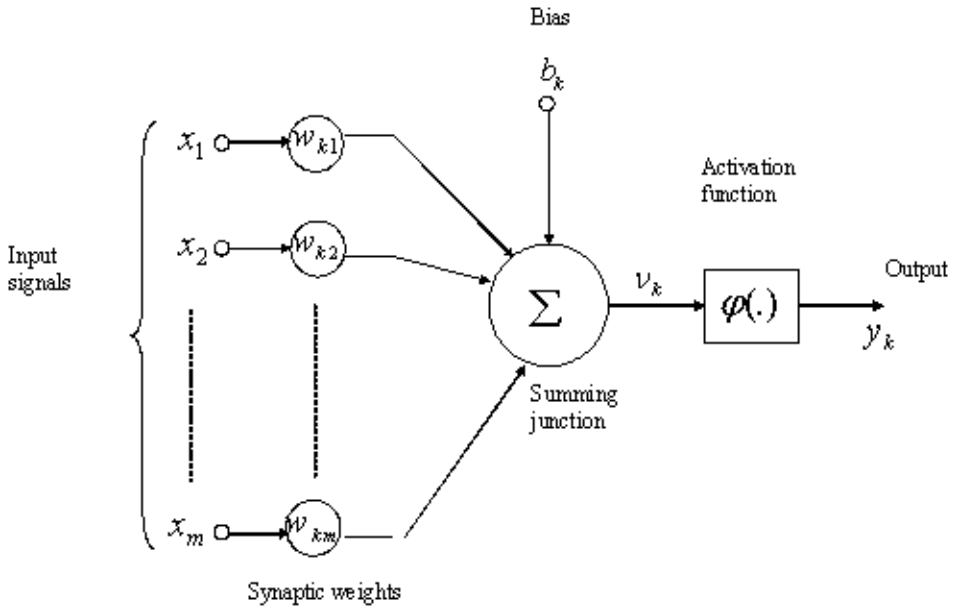

Figure 1: Non-linear model of a neuron.

A set of connecting links (synapses) are characterized by weights of its own. Signal x_j at the input of synapse j connected to the neuron k, is multiplied by a weight w_{kj}. There is an adder (linear combiner) for summing up the input signals weighted by the respective synapses of the neuron (1a). After this summation, an activation function is used for limiting the amplitude of the output of a neuron to a permissible range (1b) .

Typically, the normalized amplitude range of the output of a neuron is written as the

closed unit interval [0,1] or alternatively [-1,1].

The neuron model also includes an externally applied bias, denoted by b_k. The bias has the effect of increasing or lowering the net input of the activation function, depending on its positive or negative respectively.

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{1a}$$

$$y_k = \varphi(u_k + b_k) \tag{1b}$$

There are different types of activation functions such as threshold function, piecewise linear function, sigmoid function, etc... Sigmoid function is the most common form of activation function used in the construction of artificial neural networks. It is defined as a strictly increasing function that exhibits a graceful balance between linear and non-linear behavior [14]. It is defined by the formulation in equation (2)

$$\varphi(\nu) = \frac{1}{1 + exp(-a\nu)} \tag{2}$$

where a is the slope parameter of the sigmoid function. When slope approaches infinity the sigmoid function becomes a threshold function as seen from figure (2).

In general, we can define the neural network structure in three main sections which are input layer, hidden layer(s), output layer. The first layer is called the input layer where the information to be analyzed is fed to the neurons. Number of neurons in this layer depend on the dimension of the information that is fed to the input layer. Then this information is propagated to the neurons of the next layer (hidden layers) and the process continues until reaching the output layer.

There are different classes of network architecture.

Single-layer feedforward network is the simplest form of a layered network where we have an input layer of source nodes that projects onto an output layer neurons. There is no hidden layer in the architecture. A compact illustration is used in figure (3) where just

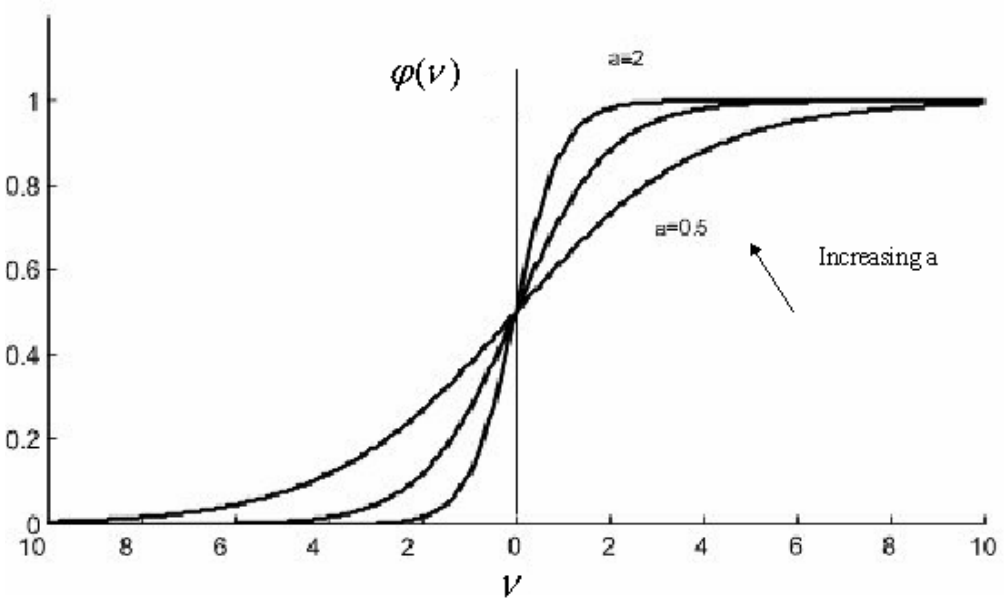

Figure 2: Sigmoid function.

an individual neuron contribution is shown. $\boldsymbol{R}$ represents the number of elements in input vector, while $\boldsymbol{S}$ represents the number of the neurons in the output layer.

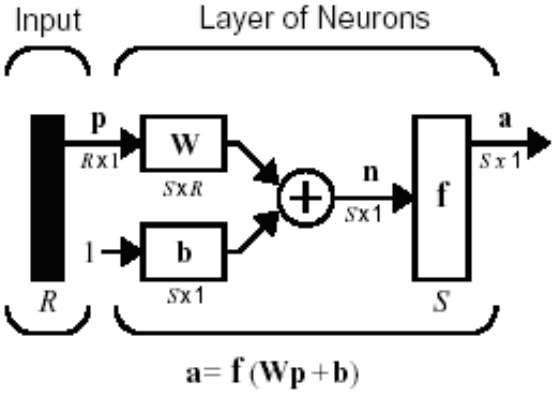

Figure 3: Single layer feedforward network.

The input vector elements enter the network through the weight matrix $\boldsymbol{W}$. Row indices on the elements indicate the destination neuron of the weight, while the column indices indicate which source is the input for that weight as seen in figure (4).

In the figure (3), $\boldsymbol{p}$ is an input vector with length $\boldsymbol{R}$, $\boldsymbol{W}$ is an $\boldsymbol{SxR}$ matrix and $\boldsymbol{a}$ and $\boldsymbol{b}$ are $\boldsymbol{S}$ length vectors. As mentioned earlier, $\boldsymbol{b}$ is the bias vector, and we also have the summer and the function operator.

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ & & & \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

Figure 4: Weight matrix.

Multi-layer feedforward network distinguishes itself by the presence of one or more hidden layers. The network is enabled to extract higher order statistics. This is important when the size of the input layer is large. Figure (5) illustrates fully connected feedforward network with one hidden layer.

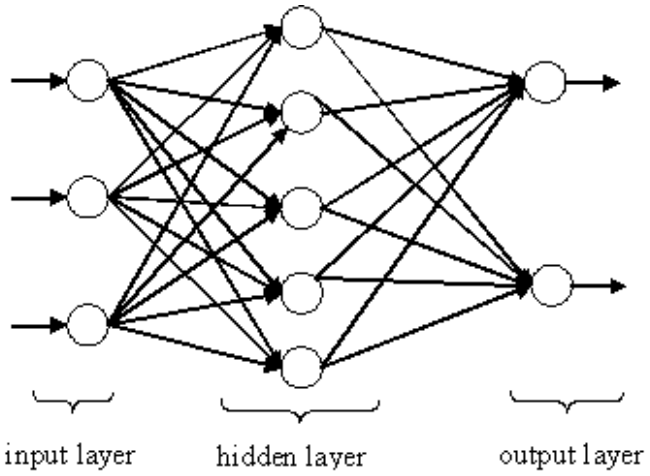

Figure 5: Fully connected 3-5-2 feedforward network structure.

Here all the nodes between subsequent layers are connected to each other. Input layer has 3 neurons, hidden layer has five neurons and output layer has two neurons (3-5-2).

In the figure (6), two hidden layer structure can be seen where layer 1 and layer 2 represents the hidden layers and layer 3 represents the output layer. Input layer is not numbered in this figure. The network has $\boldsymbol{R}$ input neurons and $\boldsymbol{S}_3$ output neurons.$\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ represents the neurons on the layer 1 and layer 2 (hidden layers) respectively. The outputs of each intermediate layer are the inputs of the next layer. IW represents the weight matrix which has a contribution with input layer and LW's represent the matrices with contribution to hidden layers and output layer.

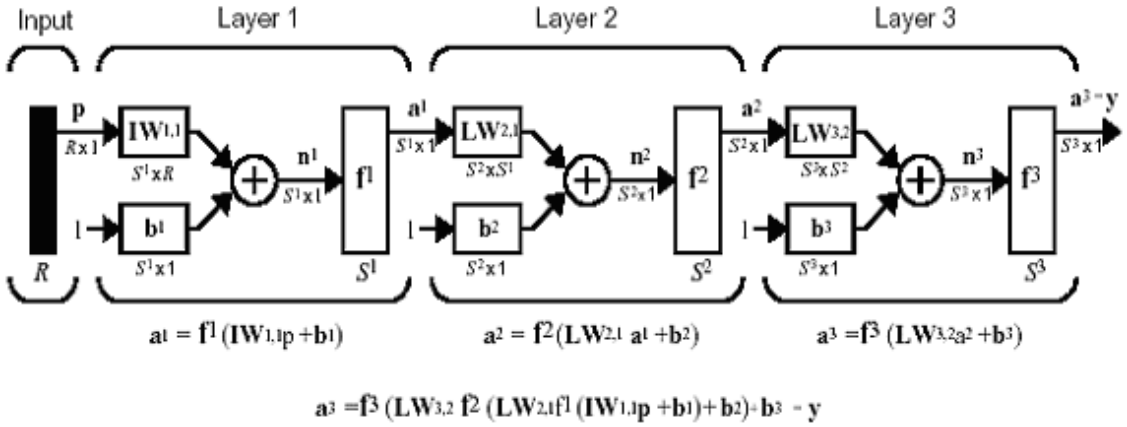

Figure 6: $\boldsymbol{R}$-$\boldsymbol{S}_1$-$\boldsymbol{S}_2$-$\boldsymbol{S}_3$ feedforward compact network illustration.

Multiple-layer networks are quite powerful. For instance, a network of three layers (one hidden layer), where the hidden is sigmoid and the output layer is linear, can be trained to approximate any function (with a finite number of discontinuities) arbitrarily well. This kind of three-layer network will be also used in the applications.

Recurrent networks distinguishes itself from feedforward neural networks in that it has at least one feedback loop.

There are some important issues that should be taken care while constructing the NN architecture:

- Decision on the number of input neurons which depends on the variety of information that is fed to the network.
- Deciding on the number of hidden layer neurons is an important and difficult task since optimal number of neurons will provide the maximum network performance.
- Selection of activation functions (non-linear or linear) according to the problem.

2.2 Training Modes (Incremental and Batch Training)

In practical applications of back-propagation algorithm, learning process results from the presentations of a prescribed set of training examples to the network. One complete presentation of the entire training set is called epoch. The learning process is maintained on an

epoch by epoch basis until the synaptic weights and bias levels of the network stabilize and the averaged squared error over the entire training set converges to some minimum value. For a given training set there can be two choices of learning ways:

1- *Incremental (Sequential) mode* : In the incremental mode of back-propagation learning, weight updating is performed after the presentation of each training example (input). Specifically, after the first example is presented to the network, it adjusts its synaptic weights and biases in order to minimize the error between the output of the network and the desired output. This is done just for one training example of the entire epoch. Then, second example is presented and new adjustments are done. This process is continued until the last example, where one epoch is completed. Then, if necessary another epoch can be processed in the same manner. This procedure is also called online or stochastic mode.

2- *Batch mode* : In this mode of back-propagation learning weight updating is performed after the presentation of all the training examples, that constitute an epoch. We can define the cost function as the averaged squared error shown in the equation (3)

$$\varepsilon_{av} = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \epsilon C} e_j^2(n) \tag{3}$$

where set of C includes all the neurons in the output layer, and the $e_j(n)$ is the error signal of the output neuron j for the iteration n, between the desired response and the corresponding response of the network. The inner summation is the error for one example and the outer summation is for the entire epoch. Clearly, in this approach we want to minimize the error for all the training examples at the same time. In other words, we want to converge to a closest local minimum for the entire set. For a learning rate of η, the adjustments applied to a synaptic weight w_{ji} connecting neuron i to neuron j is defined by the delta rule as the following:

$$\Delta w_{ji} = -\eta \frac{\partial \varepsilon_{av}}{\partial w_{ji}} \tag{4a}$$

$$\Delta w_{ji} = -\frac{\eta}{N} \sum_{n=1}^{N} e_j(n) \frac{\partial e_j(n)}{\partial w_{ji}} \tag{4b}$$

$$w_{ji_{new}} = w_{ji_{old}} + \Delta w_{ji} \tag{4c}$$

Weight adjustments are done only after the entire training set has been presented to the network.

Comparison Between Two Modes

- For the on-line training the incremental mode is preferred over the batch mode because it requires less local storage for each synaptic connection (for small training sets with few data, also batch mode can be an option for the online training as described as a part of an application in chapter(6)).

- The use of pattern by pattern updating of weights (incremental mode) makes the search in weight space stochastic in nature. This makes it less likely for the back-propagation algorithm to be trapped in local minimum.

- The stochastic nature of the incremental mode makes it difficult to establish theoretical conditions for convergence of the algorithm. However, the batch mode of training provides an accurate estimate of the gradient vector, convergence to a local minimum is thereby guaranteed under simple conditions.

As a conclusion, despite the fact that the incremental mode of back-propagation learning has several disadvantages, it is highly popular for two important practical reasons.

- The algorithm is simple to implement.

- It provides the effective solutions to large and difficult problems.

In this study, we will make use of both the batch mode and the incremental mode in an effective way for the class of problems that have been worked on.

2.3 Training Algorithms

In this chapter, several different training algorithms for feedforward networks will be discussed. The error back-propagation (EBP) algorithms which work for the feedforward structures are the most commonly used types [22]. These algorithms are widely used because of their robustness, which allows them to be applied in a wide range of tasks. Back-propagating the error from the output layer to the input layer is the way of using known input and output pairs of a target function to find the coefficients that make a certain mapping function approximate the target function as closely as possible. All of these algorithms use the gradient of the performance (cost) function to determine how to adjust the weights to minimize performance. This gradient technique is called back-propagation which involves performing computations backwards through the network.

Back-propagation Algorithm

There are many back-propagation algorithms which we discuss in this chapter. The simplest implementation of back-propagation learning, updates the network weights and biases in the direction in which the performance function decreases more rapidly (the negative of the gradient). Generally, this is called gradient descent methodology.

Before going through the different types of implementations, its better to give some information about the optimization techniques used in minimizing the cost.

Unconstrained optimization techniques

We can consider a cost $\varepsilon(\boldsymbol{w})$ function which is continuously differentiable function of some unknown vector $\boldsymbol{w}$. The aim is to find the weight (parameter) vector of an adaptive algorithm which makes the NN behave in an optimum manner. In order to do this, we have to find the optimal solution ($\boldsymbol{w}^*$) that satisfies the condition given in equation (5).

$$\varepsilon(\boldsymbol{w}^*) \leq \varepsilon(\boldsymbol{w}) \tag{5}$$

The cost function $\varepsilon(\boldsymbol{w})$ should be minimized with respect to the weight vector $\boldsymbol{w}$. Necessary condition for the optimality will be the equation (6) where ∇ is the gradient operator.

$$\boldsymbol{\nabla}\varepsilon(\boldsymbol{w}^*) = 0 \tag{6}$$

$\nabla\varepsilon(\boldsymbol{w})$ is the gradient vector of the cost function (7).

$$\boldsymbol{\nabla}\varepsilon(\boldsymbol{w}) = [\frac{\partial\varepsilon}{\partial w_1}, \frac{\partial\varepsilon}{\partial w_2}, ..., \frac{\partial\varepsilon}{\partial w_m}]^T \tag{7}$$

Starting with an initial guess for the weight vectors we can generate a sequence of weight vectors where the cost function is decreased in every iteration as shown in equation (8). However, there is no guarantee that we will eventually converge to the optimal solution $\boldsymbol{w^*}$.

$$\varepsilon(\boldsymbol{w}(n+1)) \leq \varepsilon(\boldsymbol{w}(n)) \tag{8}$$

The basic methodology used for unconstrained optimization is steepest descent.

Method of steepest descent

In this methodology, the successive adjustments applied to the weight vector $\boldsymbol{w}$ are in the direction of the steepest descent, which is in a direction opposite to the gradient vector $\nabla\varepsilon(\boldsymbol{w})$. For convenience we can take it as

$$\boldsymbol{g} = \boldsymbol{\nabla}\varepsilon(\boldsymbol{w}). \tag{9}$$

The steepest descent algorithm is described by the formula given in equation (10a) where η is a positive constant called the learning rate parameter and $\boldsymbol{g}(n)$ is the gradient vector evaluated at the point $\boldsymbol{w}(n)$. Moving from iteration n to $n+1$ the correction applied by the algorithm is given in the equation (10b).

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta \boldsymbol{g}(n) \tag{10a}$$

$$\Delta \boldsymbol{w}(n) = \boldsymbol{w}(n+1) - \boldsymbol{w}(n) \tag{10b}$$

$$\Delta \boldsymbol{w}(n) = -\eta \boldsymbol{g}(n) \tag{10c}$$

Now we can use a first order Taylor series expansion around $\boldsymbol{w}(n)$ to approximate $\varepsilon(\boldsymbol{w}(n+1))$ in order to show that the formulation satisfies the condition given in equation (8).

$$\varepsilon(\boldsymbol{w}(n+1)) \simeq \varepsilon(\boldsymbol{w}(n)) + \boldsymbol{g}^T(n)\Delta \boldsymbol{w}(n) \tag{11a}$$

$$\varepsilon(\boldsymbol{w}(n+1)) \simeq \varepsilon(\boldsymbol{w}(n)) - \eta \boldsymbol{g}^T(n)\boldsymbol{g}(n) \tag{11b}$$

$$\varepsilon(\boldsymbol{w}(n+1)) \simeq \varepsilon(\boldsymbol{w}(n)) - \eta \parallel \boldsymbol{g}(n) \parallel^2 \tag{11c}$$

Equation (11c) shows that for a positive learning rate η, the cost function is decreased as the algorithm progresses from one iteration to next. The reasoning presented here is only true for small learning rates.

The method of steepest descent converges to the optimal solution $\boldsymbol{w}^*$ slowly. Furthermore, the learning rate has a profound influence on the convergence. In this part, it will be useful to talk about the importance of the learning rate.

Learning Rate

The learning rate is one of the most important parameters in the steepest descent methodology. In the equation (10c), η represents the learning rate which is a positive value between 0 and 1. Learning rate has the same value for each connection (weight) where it adjusts the step size that will be taken along the line corresponding to the steepest gradient. It is downwards at the current weight state along the error surface over the weight space [23]. Generally in standard back-propagation η is also kept fixed throughout the application. When learning rate kept constant, the length of the steps will be in a

fixed proportion to the size of the steepest gradient. Behavior resulting from this feature will be most successful for error surfaces with large initial steepest gradients that become shallow near the goal weight state. The initial largeness reduces the number of steps needed to cover the initial ground towards the goal. However, large values for learning rate can cause overshooting the goal (i.e. global error) by taking large steps. As a simplest case in figure (7), we can demonstrate the gradient descent in one dimension (respect to one weight). As it is discussed above, learning rate has to be chosen reasonably small enough in order not to overshoot the goal.

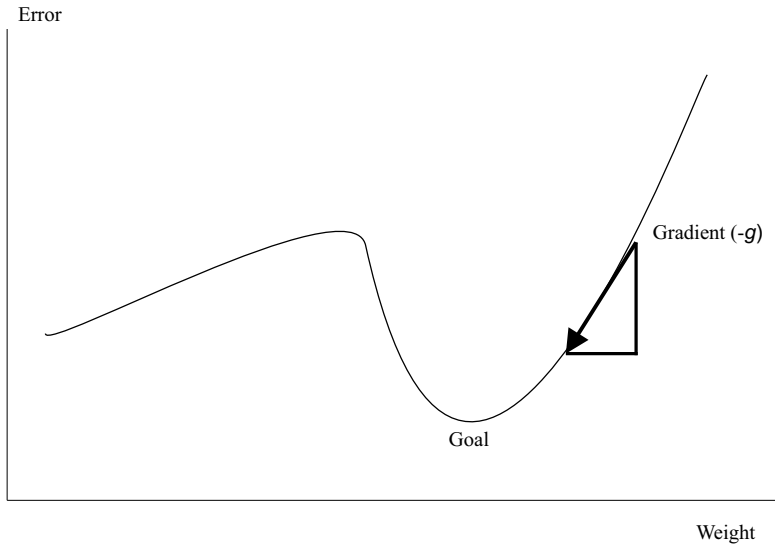

Figure 7: Gradient descent in one dimension.

In the aspect of the weight changes, it can be said that the smaller we make the learning rates, the smaller the changes to the synaptic weights in the network will be from one iteration to next. This will cause a smoother trajectory in the weight space. This improvement is attained at the cost of a slower learning rate. On the other hand, if we increase the learning rate to speed up the process network may become unstable (i.e., oscillatory) in terms of weights.

There is a simple application in the reference [14] in order to show the effect of the changes in the learning rates to the algorithm. Due to these results, we can make some interpretation as follows:

- When η is small the transient response of the algorithm is overdamped, the trajectory

traced by $\boldsymbol{w}$ has a smooth path. For this case, the algorithm will take too long to converge.

- When η is large the transient response of the algorithm is underdamped, trajectory $\boldsymbol{w}$ follows has a oscillatory path.

- When η exceeds a certain critical value, the algorithm becomes unstable.

As a conclusion, we can say that the performance of the algorithm is very sensitive to the proper setting of the learning rate.

There are two different ways in which the steepest descent algorithm can be implemented. These are incremental and batch modes that have been discussed in the previous section.

Steepest descent with Momentum rate

A well-known augmentation to back-propagation which speeds up travel over shallow surfaces is to use momentum coefficient α to allow previous weight change to have continuing influence on the current weight change. Formulation of this method is presented in equation (12) where α is set between 0 and 1.

$$\Delta \boldsymbol{w}_{ji}(n) = -\eta \frac{\partial \varepsilon}{\partial \boldsymbol{w}_{ji}} + \alpha \Delta \boldsymbol{w}_{ji}(n-1) \tag{12}$$

Momentum rate allows a network to respond not only to the local gradient but also to recent trends in the error surface. It acts like a low-pass filter and allows the network to ignore small features in the error surface. The use of momentum rate has both an accelerating effects, where the current negative of the error-weight derivative has the same direction to the previous weight change $\Delta \boldsymbol{w}_{ji}(n-1)$, and a damping effect, where the terms are opposite in sign.

The performance of the steepest descent algorithm can also be improved if the learning rate is allowed to be changed during the process. The use of the adaptive learning rate with the steepest descent methodology is discussed in detail in the chapter(6).

There are some other augmentations that improve the performance of the steepest descent methodology. For example, a sigmoid function can cause the gradients to have a very small magnitude since if the inputs get large in value slope of the sigmoid function goes to zero. Small changes in gradients will cause small changes in the weights and biases even though they are far from their optimal values. This problem can be overcome by using *resilient back-propagation algorithm* where these harmful effects of the magnitudes of the partial derivatives can be eliminated. In this algorithm, only the sign of the derivative is used to determine the direction of the weight update, the magnitude of the derivative has no effect on the update.

Furthermore, there are some *conjugate gradient algorithms* where a search is performed along conjugate directions rather than the steepest descent directions to provide faster convergence. In this case, the step size is adjusted in each iteration.

Newton's Method

This methodology is based on minimizing the quadratic approximation of the cost function $\varepsilon(\boldsymbol{w})$ around the current point $\boldsymbol{w}(n)$. This minimization is performed at each iteration of the algorithm. It uses a second order Taylor series expansion of the cost function around the point $\boldsymbol{w}(n)$ as seen in equation (13).

$$\Delta\varepsilon(\boldsymbol{w}(n)) = \varepsilon(\boldsymbol{w}(n+1)) - \varepsilon(\boldsymbol{w}(n)) \tag{13a}$$

$$\Delta\varepsilon(\boldsymbol{w}(n)) \simeq \boldsymbol{g}^T(n)\Delta\boldsymbol{w}(n) + \frac{1}{2}\Delta\boldsymbol{w}^T(n)\boldsymbol{H}(n)\Delta\boldsymbol{w}(n) \tag{13b}$$

For the steepest descent case $\boldsymbol{g}(n)$ is the m-by-1 gradient vector of the cost function $\varepsilon(\boldsymbol{w})$ evaluated at the point $\boldsymbol{w}(n)$. In this case, the matrix $\boldsymbol{H}(n)$ is the m-by-m *Hessian matrix* of $\varepsilon(\boldsymbol{w})$ also evaluated at $\boldsymbol{w}(n)$. The Hessian matrix is defined by the equation (14).

$$\boldsymbol{H} = \nabla^2 \varepsilon(\boldsymbol{w}) \tag{14a}$$

$$\boldsymbol{H} = \begin{bmatrix} \frac{\partial^2 \varepsilon}{\partial w_1^2} & \frac{\partial^2 \varepsilon}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 \varepsilon}{\partial w_1 \partial w_m} \\ \frac{\partial^2 \varepsilon}{\partial w_2 \partial w_1} & \frac{\partial^2 \varepsilon}{\partial w_2^2} & \cdots & \frac{\partial^2 \varepsilon}{\partial w_2 \partial w_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 \varepsilon}{\partial w_m \partial w_1} & \frac{\partial^2 \varepsilon}{\partial w_m \partial w_2} & \cdots & \frac{\partial^2 \varepsilon}{\partial w_m^2} \end{bmatrix} \tag{14b}$$

In this case, the cost function $\varepsilon(\boldsymbol{w})$ is required to be twice continuously differentiable with respect to the elements of $\boldsymbol{w}$. When we differentiate the equation (13b) with respect to $\Delta\boldsymbol{w}$, we will end up with the equation (15) where the change $\Delta\varepsilon(\boldsymbol{w})$ is minimized. Solving this equation for $\Delta\boldsymbol{w}$, will lead to the formulation that we can use to adjust the weights of the neural network.

$$\boldsymbol{g}(n) + \boldsymbol{H}(n)\Delta\boldsymbol{w}(n) = 0 \tag{15}$$

Newton's method converges faster than the other gradient algorithms that are mentioned. Also it does not exhibit the zigzagging behavior that the steepest descent sometimes shows. However, the Hessian matrix has to be a positive definite matrix for all n's and there is no guarantee that $\boldsymbol{H}(n)$ is positive definite for each iteration. In this case, some modifications are necessary to this method [14].

Levenberg-Marquardt Method

This method is also designed to approach second-order training speed like the Newton method. However, there is no need to compute the Hessian matrix. When the performance function has the form of a sum of squares, then the Hessian matrix can be approximated as $H = J^T J$ and the gradient can be computed as $\boldsymbol{g} = J^T e$ where J is the Jacobian matrix which contains the first derivatives of the network errors with respect to the weights and biases, and e is a vector of network errors as depicted in the equations (16).

$$\boldsymbol{e}(n) = \begin{bmatrix} e(1), & e(2), & \ldots, & e(n) \end{bmatrix}^T \tag{16a}$$

$$\boldsymbol{J}(n) = \begin{bmatrix} \frac{\partial e(1)}{\partial w_1} & \frac{\partial e(1)}{\partial w_2} & \cdots & \frac{\partial e(1)}{\partial w_m} \\ \frac{\partial e(2)}{\partial w_1} & \frac{\partial e(2)}{\partial w_2} & \cdots & \frac{\partial e(2)}{\partial w_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e(n)}{\partial w_1} & \frac{\partial e(n)}{\partial w_2} & \cdots & \frac{\partial e(n)}{\partial w_m} \end{bmatrix}_{\boldsymbol{w}=\boldsymbol{w}(n)} \tag{16b}$$

The n-by-m Jacobian matrix can be computed through a standard back-propagation technique that is less complex than computing the Hessian matrix. This Levenberg-Marquardt methodology uses the following approximated Hessian matrix in the Newton-like update of the weights.

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - [J^T J + \mu I] J^T e \tag{17}$$

In the equation (17), when the scalar μ is zero, this is just Newton's method using the approximate Hessian matrix. When μ is large, this becomes gradient descent with a small step size. Newton's method is faster and more accurate near an error minimum, so the aim is shift towards Newton's method from gradient descent as quickly as possible. In order to do this, μ is decreased after each successful step (reduction in performance function) and increased only when there is an increase in the performance function [24] .

Among all these methodologies Levenberg-Marquardt algorithm appears to be the fastest method for training moderate-sized feedforward neural networks (up to several hundred weights). Although the performance of this methodology is generally satisfactory in complex problems with batch mode, steepest descent is mostly preferred for on-line training applications because of its simplicity in implementation to the code and in control of the parameter behaviors (i.e. learning rate), and its convenience in working with limited data.

Back-propagation Algorithm for Multilayer Networks

In order to be clear defining the process, lets define the error signal at the output layer

and go back through the input layer by back-propagating the error. Equation (18) stands for the error at the output neuron j for the iteration n (example n) where right hand side is the difference between the desired output and the network output.

$$e_j(n) = d_j(n) - y_j(n) \tag{18}$$

The total error is obtained by summation over all neurons in the output layer where C includes all the neurons in the output layer of the network in equation (19). This is the total error for only one example (not complete set) where it can be used for the incremental (on-line) training with one example each step.

$$\varepsilon(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n) \tag{19}$$

Error $\varepsilon(n)$ represents the performance function that we want to decrease applying the gradient descent methodology. Equation (20) defines the correction for specific weighting factor w_{ji} by the delta rule.

$$\Delta w_{ji}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} \tag{20}$$

The partial derivative $\partial\varepsilon(n)/\partial w_{ji}(n)$ represents a sensitivity factor determining the direction of the search in weight space for the synaptic weight w_{ji}, while η represents the learning rate of the back-propagation algorithm. The use of the minus sign stands for the gradient descent in weight space (seeking a direction for weight change that reduces the $\varepsilon(n)$).

Equation (20) can also be defined in the following way,

$$\Delta w_{ji}(n) = -\eta \delta_j(n) y_i(n) \tag{21}$$

where local gradient $\delta_j(n)$ and input signal of neuron j $y_i(n)$ is related with the equation (22) and equation (23) respectively. Signal flow graph (8) for an individual neuron is also included to make the interactions clear.

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} \tag{22a}$$

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \tag{22b}$$

$$y_i(n) = \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{23}$$

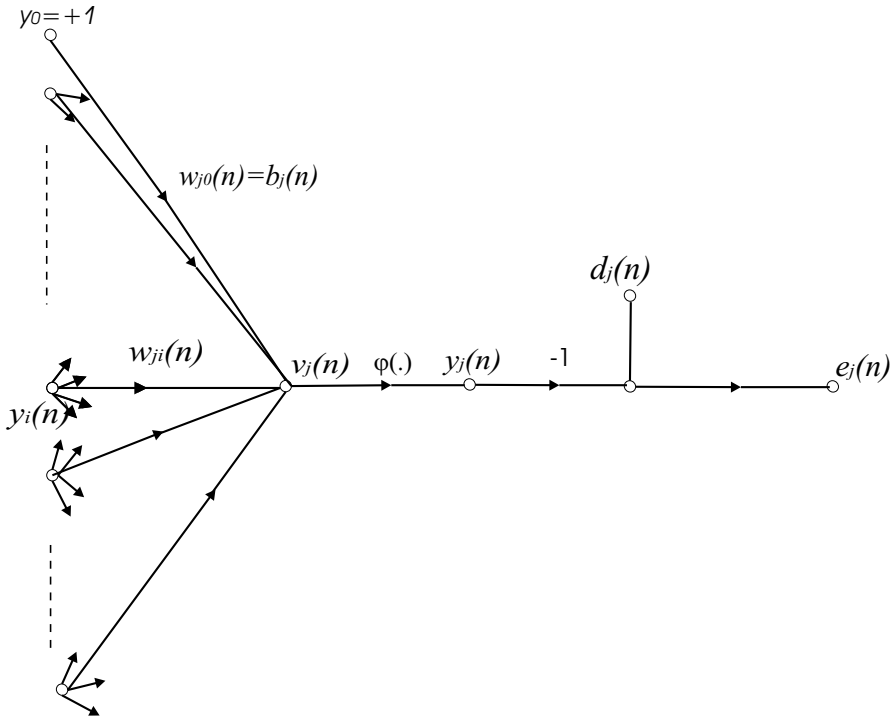

Figure 8: Signal-flow graph for the details of output neuron j.

2.4 System Identification and Adaption

The nonlinear functional mapping properties of neural networks are central to their use in control. Training a neural network using input-output data from a plant can be considered as a nonlinear functional approximation problem. Identification can be done with either forward modelling or inverse modelling.

The procedure of training a neural network to represent the forward dynamics of a system will be referred to as forward modelling. In general, system identification describe

the input-output relation of an unknown multiple input-multiple output (MIMO) systems. A structure for achieving this relation is shown schematically in figure (9). The neural network model is placed parallel with the system and the prediction error is used as the network training signal. To make it clear, lets implement an input vector $\mathbf{x}_i$ to both an unknown system and a NN based model. As shown in figure (9), the output of NN is denoted by $\mathbf{y}_i$ and the output of the unknown system (black box) is $\mathbf{d}_i$. The difference between the $\mathbf{d}_i$ and the network output $\mathbf{y}_i$ provides the error signal vector $\mathbf{e}_i$. This error signal is then used to adjust the free parameters of the network to minimize the squared difference between the outputs of the unknown system and the neural network in a statistical sense and, is computed over the entire training set [14, 20].

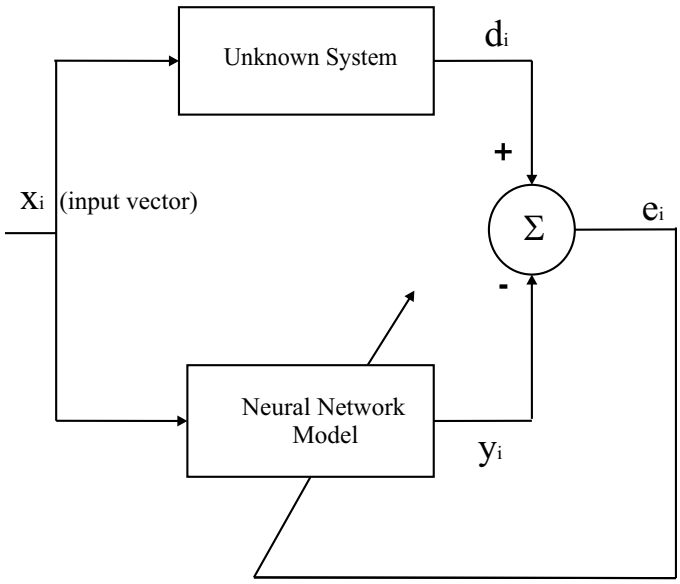

Figure 9: Block diagram of system identification.

Also another way to approximate an unknown input-output mapping can be using an inverse system methodology [14, 20].

The control of a plant is an important learning task that can be done by a neural networks. Adaption of the controller takes a vital role in order to provide improved stability of the plant.

However, the environment of interest is frequently non-stationary where the parameters generated by the environment vary with time. In this situations, the traditional methods

of learning will be inadequate since the network will not be capable to track the statistical variations of the environment in which it operates. To overcome this problem, it is desirable for a neural network to continually adapt its free parameters to variations in the incoming information in a real-time. The learning process encountered in an adaptive system never stops, it continues with the new information processed by the system. This type of learning is called *continuous learning* or *learning-on-the-fly* [14].

In the continuous learning neural network is able to adapt its behavior to the varying temporal structure of the incoming signal since statistical characteristics of a non-stationary process usually change slowly enough for the process to be considered as pseudo-stationary over a window of short enough duration. This pseudo-stationary property of a stochastic process can be used to extend the utility of a neural network by retraining it at some regular intervals to account for statistical fluctuations of the incoming data.

For a more refined dynamic approach to learning, following steps can be used:

- Select a window short enough for the input data to be considered pseudo-stationary and use the data to train the network.
- When a new data sample is received update the data window and use this to retrain the network.
- Repeat the procedure on a continuing basis.

Direct and Indirect Control

The control of a *plant* is another learning task that can be done by a neural network. The plant is the process or critical part of the system that is to be maintained in a controlled condition. Mainly, two different approaches have been used to control a plant adaptively for over 20 years. These are *direct* and *indirect control*. In direct control, the parameters of the controller are directly adjusted to reduce some norm of the output error. In indirect control, the parameters of the plant are estimated as the elements of a vector $\hat{\boldsymbol{p}}_{\boldsymbol{p}}(n)$ at any instant n and the parameter vector $\boldsymbol{\theta}_c(n)$ of the controller is chosen assuming that $\hat{\boldsymbol{p}}_{\boldsymbol{p}}(n)$ represents the true value $\boldsymbol{p}_{\boldsymbol{p}}$ of the plant parameter vector. Since it can be shown that

controller parameter vector $\boldsymbol{\theta}_c^*(n)$ exist for every value of the plant parameter vector $\boldsymbol{p_p}$, so that the output of the controlled plant together with the controller approaches the output of the reference model asymptotically [18]. Even when the plant is assumed to be linear and time variant, both direct and indirect adaptive control results in overall non-linear systems [18]. Figures (10) and (11) represent the structure of the overall adaptive system using the two methods for the adaptive control of a linear time-invariant plant [18, 19].

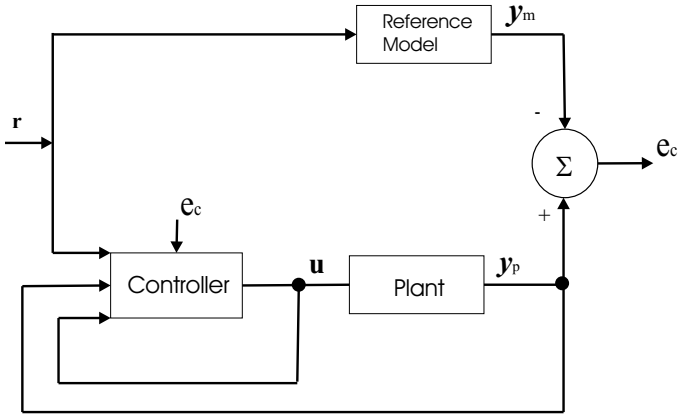

Figure 10: Direct adaptive control

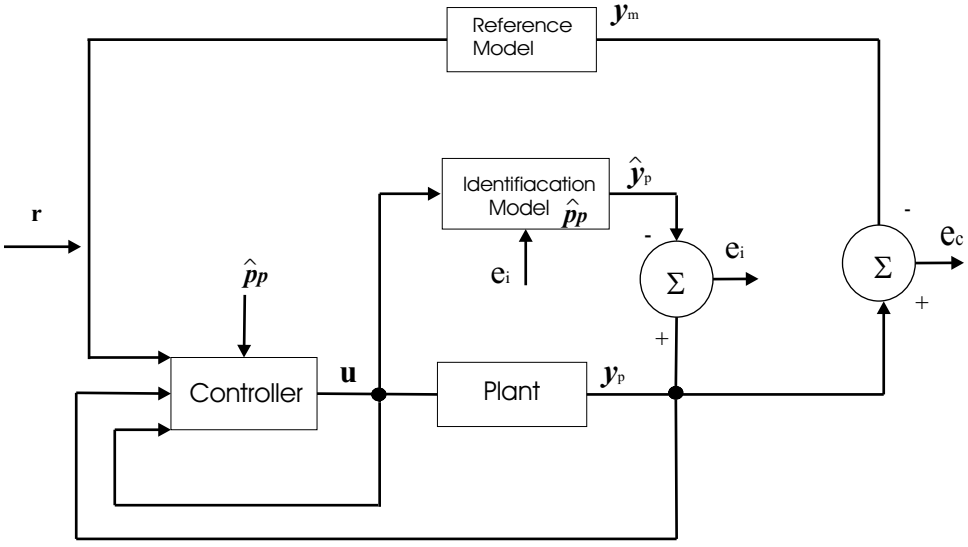

Figure 11: Indirect adaptive control

At present, methods for directly adjusting the control parameters based on the output error (between the plant and reference model outputs) are not available. This is because the unknown nonlinear plant in Figure (10) lies between the controller and the output

error $\mathbf{e}_c$. Hence, until such methods are developed, adaptive control of nonlinear plants has to be carried out using indirect methods. Using the resulting identification model, which contains neural networks as a controller and linear dynamical elements as subsystems, the parameters of the controller are adjusted. The identification model can be used to compute the partial derivatives of a performance index with respect to the control parameters.

Adaptively control a nonlinear plant depends largely on the prior information available regarding the unknown plant. This includes knowledge of the number of equilibrium states of the unforced system, their stability properties, as well as the amplitude of the input for which the output is bounded. For example, if the plant is known to have a bounded output for all inputs u belonging to same compact set U, then the plant can be identified off-line. During the identification the weights in the identification model can be adjusted at every instant of time or at discrete time intervals. Once the plant has been identified to the desired level of accuracy, control action can be initiated so that the output of the plant follows the output of a stable reference model. It must be emphasized that even if the plant has bounded outputs for bounded inputs, feedback control may result in unbounded solutions. Hence, for on-line control, identification and control must proceed simultaneously.

Predictive control

In the realm of optimal and predictive control methods the receding horizon technique has been introduced as a natural, computationally feasible feedback law. It has been proven that the method has a desirable stability properties for nonlinear systems.

In this approach a neural network model provides prediction of the future plant response over the specified horizon (12). The predictions supplied by the network are passed to a numerical optimization routine which attempts to minimize a specified performance criterion in the calculation of a suitable control signal.

The control signal $\acute{u}$ is chosen to minimize the quadratic performance criterion subject to the constraints of the dynamical model.

Another possibility illustrated in the figure (12), is to train a further network to mimic the action of the optimization routine. This controller network is trained to produce the

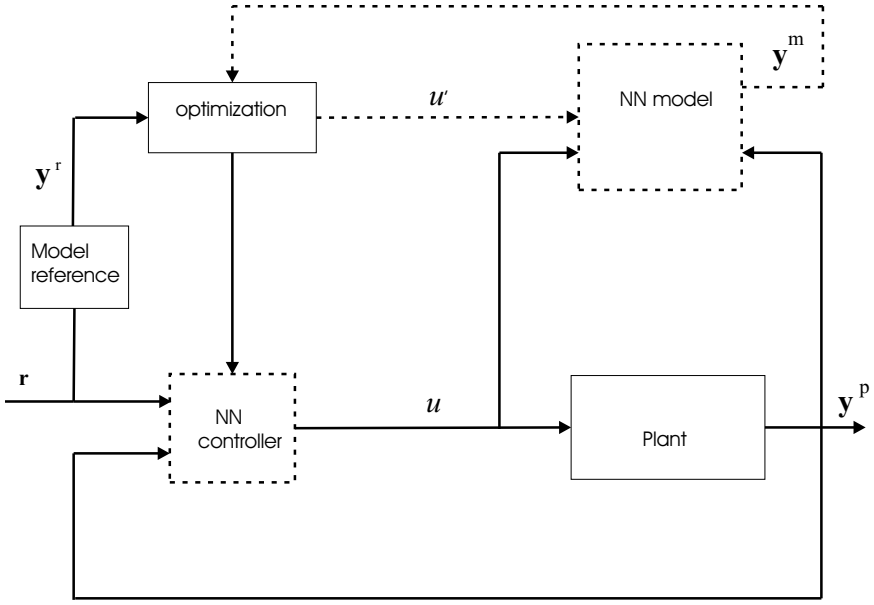

Figure 12: Structure for predictive control

same control output u, for a given plant output, as the optimization routine $\acute{u}$. An advantage of this approach is that the outer loop consisting of plant model and optimization routine is no longer needed when training is complete [20].

Model predictive controller (MPC) is a control algorithm which solves an optimization problem on-line at each time step. For problems adequately described by linear models, the linear MPC algorithm is an efficient algorithm which incorporates inherent multivariable and constraint handling capabilities. In some cases, however, the selection of the desired operating range coupled with possibly nonlinear process dynamics can degrade performance and potentially destabilize the closed-loop system. In that case, the *nonlinear model predictive control (NMPC)* algorithm is a powerful control technique that can alleviate this performance degradation while retaining the multivariable and constraint handling benefits of MPC algorithm. Control using nonlinear models can be further complicated when working with distributed parameters. Also efficient solution techniques for NMPC problems are necessary when solution time or constraints are important. For systems where incomplete information is available, the estimation analogue of NMPC, nonlinear moving horizon estimation, can be incorporated into the algorithm [20]. Detailed description of the NMPC methodology is provided in chapter(4).

CHAPTER III

DIFFERENT APPROACHES IN IDENTIFICATION OF SYSTEM ERRORS

The methodology that is proposed in the introduction is based on the NMPC methodology. By the nature of this control methodology, we will have a full model which will be considered as a plant and we will also have a nonlinear controller which is represented by the receding horizon model predictive method solved by a direct transcription approach. This method uses a reduced model augmented by a neural network.

In this section, two different approaches are proposed in order to capture the defect of the reduced model or to estimate the state derivative errors between two models. Performances of both approaches will be evaluated by the use of an application of a simple dynamic problem. This simple example is used for demonstrating the correct implementation of both models and the neural network to the methodology.

Lets define both approaches by using a reduced model $\mathcal{M}$ and full model $\widetilde{\mathcal{M}}$. The goal is matching the outputs of those two models ($\boldsymbol{y} \approx \widetilde{\boldsymbol{y}}$).

In the first approach, the error captured by the neural network can be defined as a function of reduced model states $\boldsymbol{y}$, full model states $\widetilde{\boldsymbol{y}}$ and some control inputs $\boldsymbol{u}$. The error between the two models formulated as in equation (24d). Clearly, in this approach, the error is computed between full and reduced models where the model equations are functions of their corresponding states.

$$\dot{\boldsymbol{y}} - \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{u}) = 0 \tag{24a}$$

$$\dot{\widetilde{\boldsymbol{y}}} - \widetilde{\boldsymbol{f}}(\widetilde{\boldsymbol{y}}, \boldsymbol{u}) = 0 \tag{24b}$$

$$\boldsymbol{\epsilon}(\widetilde{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{u}) = \dot{\widetilde{\boldsymbol{y}}} - \dot{\boldsymbol{y}} \tag{24c}$$

$$\boldsymbol{\epsilon}(\widetilde{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{u}) = \widetilde{\boldsymbol{f}}(\widetilde{\boldsymbol{y}}, \boldsymbol{u}) - \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{u}) \tag{24d}$$

In the second approach, the defect captured by the neural network can be defined as a function of full model states $\widetilde{\boldsymbol{y}}$, derivative of full model states $\dot{\widetilde{\boldsymbol{y}}}$ and some control inputs $\boldsymbol{u}$. Formulation of the defect can be seen from the equation (25d). Since we are trying to ensure the matching of reduced and full model states, we can use the equation (25c) on the (25b) equation to come up with the defect formulation. Clearly, in this approach, the defect is computed between full and reduced models where the model equations are both functions of full model states $\widetilde{\boldsymbol{y}}$.

Specifically, approach-2 requires the implementation of the full model states to the reduced model equations in order to evaluate the defect.

$$\dot{\widetilde{\boldsymbol{y}}} - \widetilde{\boldsymbol{f}}(\widetilde{\boldsymbol{y}}, \boldsymbol{u}) = 0 \tag{25a}$$

$$\dot{\boldsymbol{y}} - \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{u}) - \boldsymbol{d}(\dot{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{u}) = 0 \tag{25b}$$

$$\boldsymbol{y} = \widetilde{\boldsymbol{y}} \tag{25c}$$

$$\boldsymbol{d}(\dot{\widetilde{\boldsymbol{y}}}, \widetilde{\boldsymbol{y}}, \boldsymbol{u}) = \widetilde{\boldsymbol{f}}(\widetilde{\boldsymbol{y}}, \boldsymbol{u}) - \boldsymbol{f}(\widetilde{\boldsymbol{y}}, \boldsymbol{u}) \tag{25d}$$

A problem with two degrees of freedom seen in figure (13) is chosen for this example. A pendulum of length L and mass m mounted on a cart of mass M which is connected to the ground by means of spring stiffness constant k and a dashpot of constant c. The problem is represented by two generalized coordinates. The displacement of the cart and the stretch of spring is denoted by x and the angular deflection of the pendulum with respect to the vertical is denoted by θ.

A real model (plant) and a simplified model (reduced model) are defined to make the problem similar to our main objective.

Real model is defined by the dynamical equations and the parameters given in equations (26).

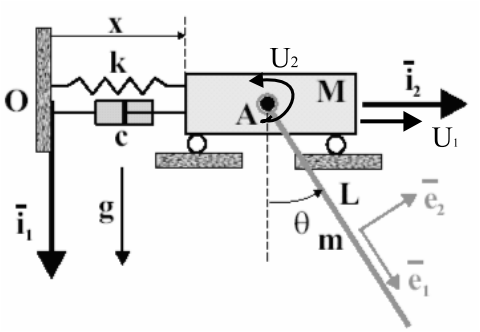

Figure 13: Dynamic system with two degrees of freedom.

$$(M+m)\ddot{x} + \frac{mL}{2}\cos\theta\,\ddot{\theta} - \frac{mL}{2}\sin\theta\,\dot{\theta}^2 + kx + c\dot{x} = \mathrm{U}_1 \tag{26a}$$

$$\frac{mL}{2}\cos\theta\,\ddot{x} + \frac{mL^2}{3}\,\ddot{\theta} + mg\frac{L}{2}\sin\theta = \mathrm{U}_2 \tag{26b}$$

$$M = 5\ kg. \qquad m = 2\ kg. \qquad L = 0.4\ m. \qquad k = 10\ N/m \qquad c = 0.5\ N\,sec/m \tag{26c}$$

In the simplified model coupled terms are disregarded. The simple dynamic equations can be seen below.

$$M\ddot{x} + kx + c\dot{x} = \mathrm{U}_1 \tag{27a}$$

$$\frac{mL^2}{3}\,\ddot{\theta} + mg\frac{L}{2}\sin\theta = \mathrm{U}_2 \tag{27b}$$

Applied controls U_1 and U_2 are selected as harmonic functions depicted in equation (28) where a_1, b_1, a_2, b_2 are constants and def_1, def_2 are the deflections used to expand the controls for training. The results with the approach-1 and approach-2 can be seen from the figures given between pages 34 to 37.

$$\mathrm{U}_1 = a_1 e^{-t}\sin(b_1\,t) \pm def_1 \tag{28a}$$

$$\mathrm{U}_2 = a_2 e^{-t}\sin(b_2\,t) \pm def_2 \tag{28b}$$

When we compare the results for the different approaches, it is obvious that approach-2 promising better performance than the approach-1. In this manner, all the examples that we are dealing with will be based on approach-2.

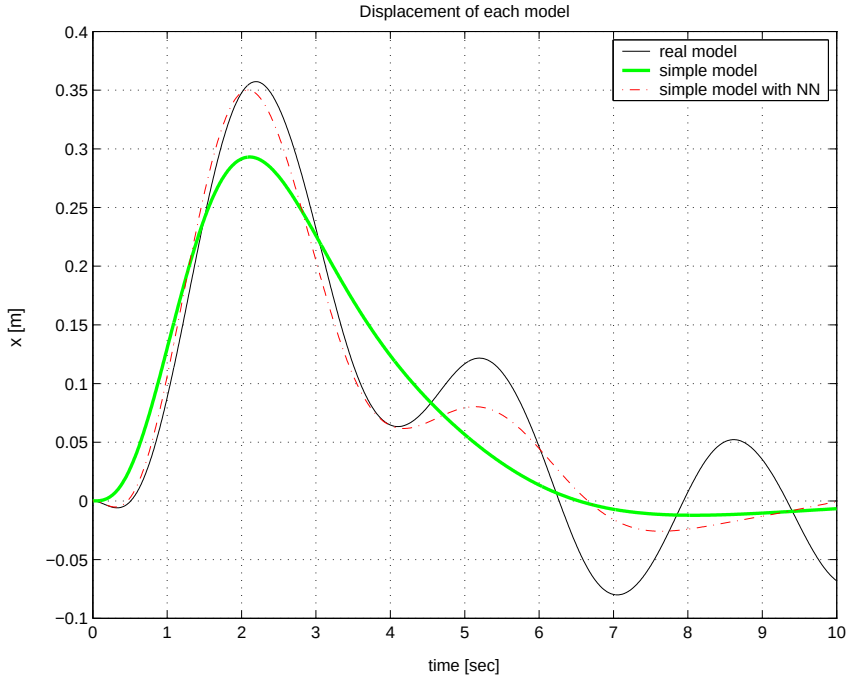

Figure 14: Displacement of the cart with approach 1

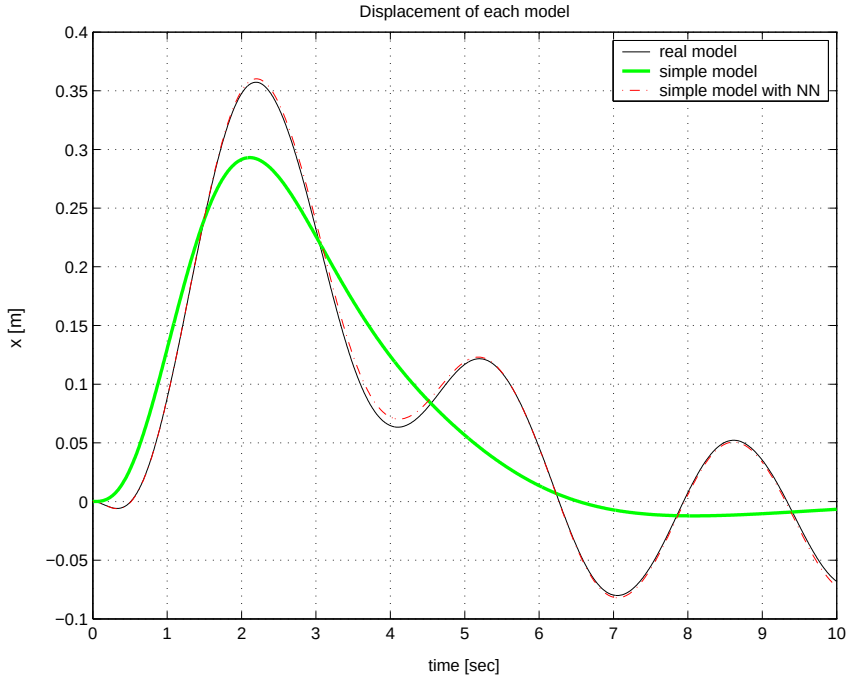

Figure 15: Displacement of the cart with approach 2

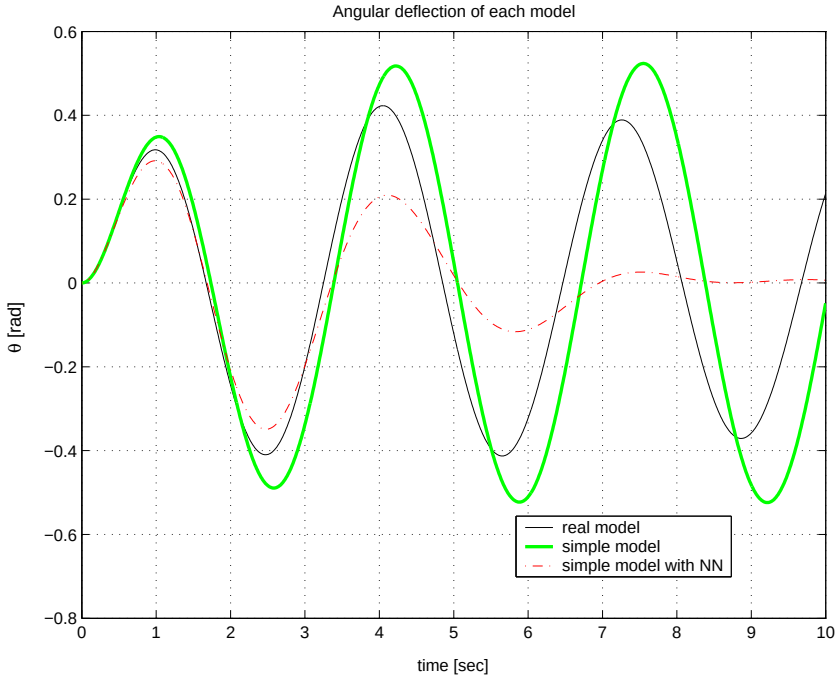

Figure 16: Angular deflection of the pendulum with approach 1

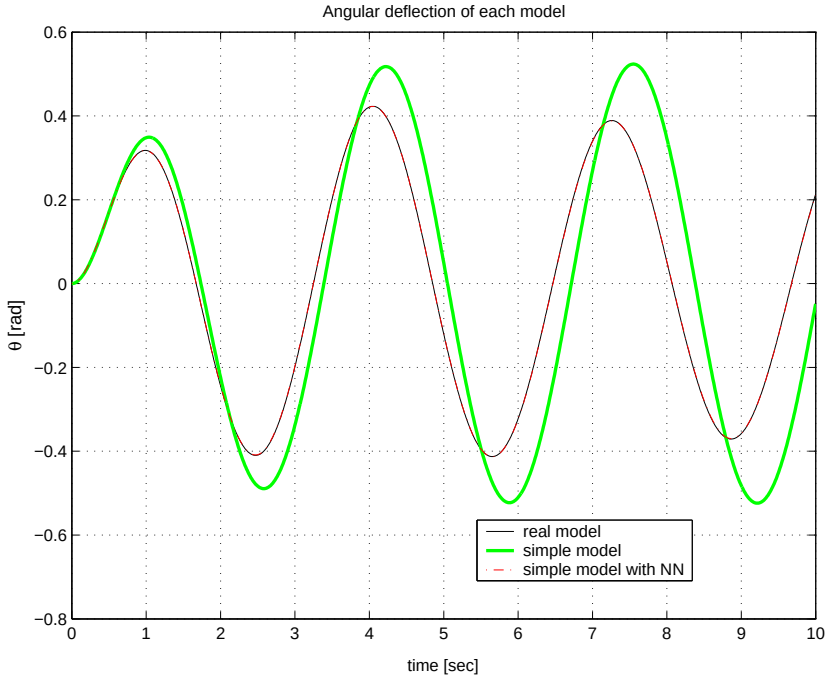

Figure 17: Angular deflection of the pendulum with approach 2

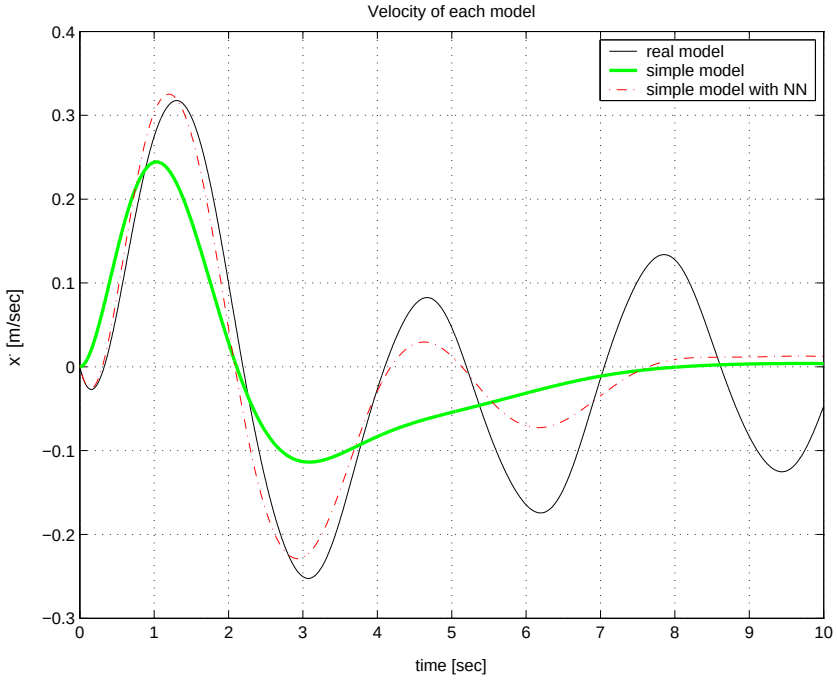

Figure 18: Horizontal velocity of the cart with approach 1

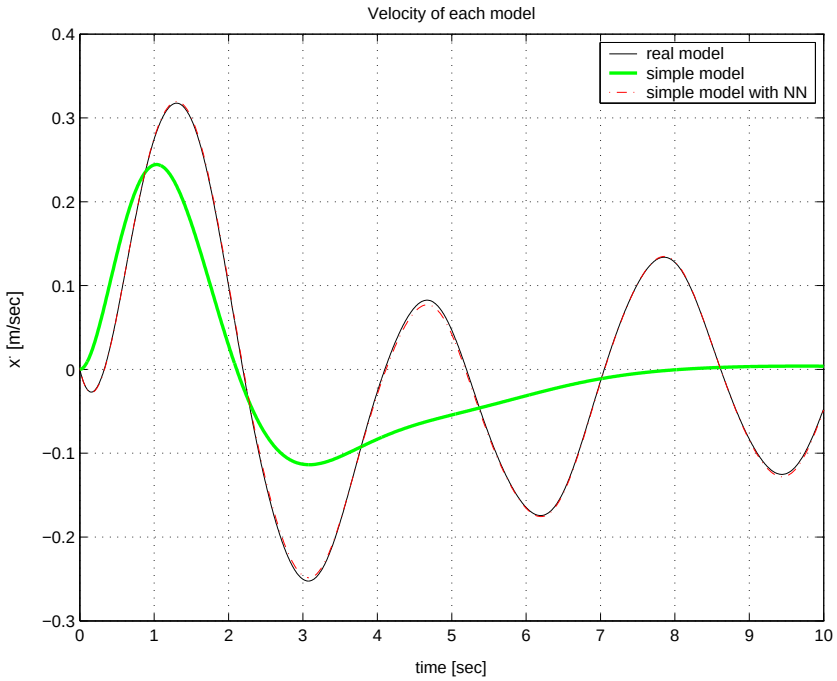

Figure 19: Horizontal velocity of the cart with approach 2

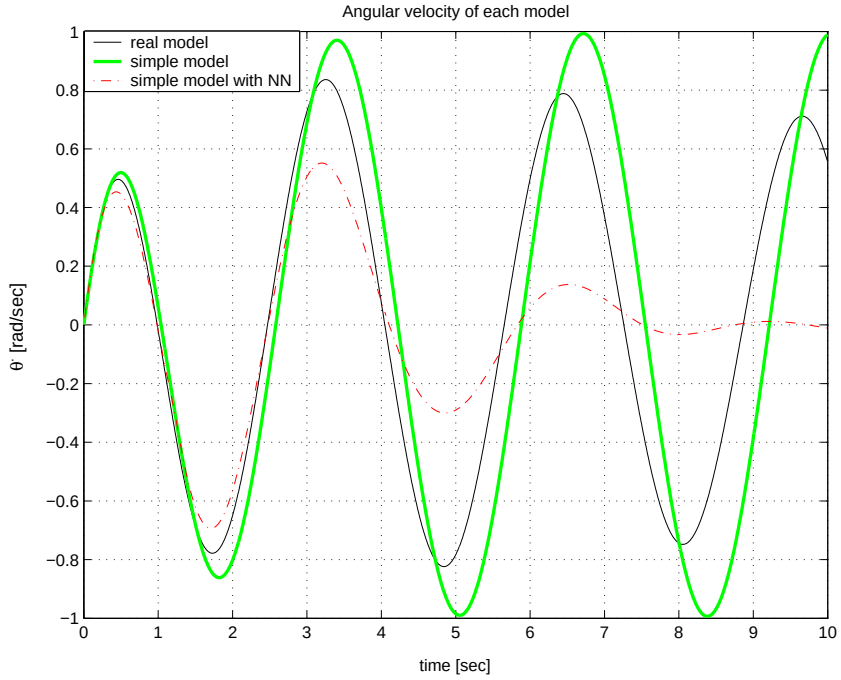

Figure 20: Angular velocity of the pendulum with approach 1

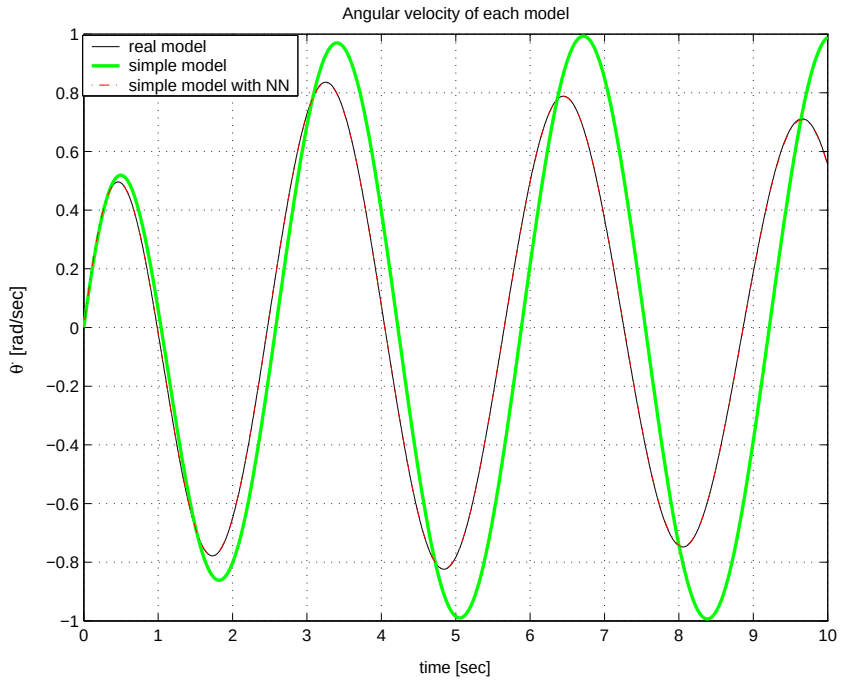

Figure 21: Angular velocity of the pendulum with approach 2

CHAPTER IV

ADAPTIVE TRACKING AND STEERING MULTIBODY CASES

In general, a multibody model characterized by $\widetilde{\mathcal{M}}$, can include rigid and flexible bodies, sensors, actuators, point elements, controls and interactional forces with environment [1, 6]. Governing system of differential-algebraic (DAE) equations of $\widetilde{\mathcal{M}}$ are defined by (29a) and (29b), where (29a) stands for the kinematic and dynamic equilibrium equations and (29b) represents the holonomic and non-holonomic constraints.

$$\widetilde{\boldsymbol{f}}(\dot{\widetilde{\boldsymbol{x}}}, \widetilde{\boldsymbol{x}}, \widetilde{\boldsymbol{\lambda}}, \widetilde{\boldsymbol{u}}) + \boldsymbol{f}_A\left(\boldsymbol{x}_A,\ \widetilde{\boldsymbol{x}}(\tau),\ \tau \in (-\infty, t)\right) = 0 \tag{29a}$$

$$\widetilde{\boldsymbol{c}}(\dot{\widetilde{\boldsymbol{x}}}, \widetilde{\boldsymbol{x}}) = 0 \tag{29b}$$

In the equations depicted above, the states of the multibody system are represented by $\widetilde{\boldsymbol{x}}$, the Lagrange multipliers which enforce the constraints are denoted by $\widetilde{\boldsymbol{\lambda}}$ and the controls are defined by $\widetilde{\boldsymbol{u}}$. These controls may represent applied forces, actuator inputs, joint relative displacements and rotations. We have to discretize the governing equations if we have flexible components in our model. In this case, because of the discretization, the degrees of freedom of the states $\widetilde{\boldsymbol{x}}$ will depend on the spatial grids of the flexible components.

Also, in the equation (29a), $\widetilde{\boldsymbol{f}}$ includes inertial, internal, external forces for the multibody system, while $\boldsymbol{f}_A$ represents the aerodynamic forces and the $\boldsymbol{x}_A$ aerodynamic states. Multibody models are based on the finite element multibody approach which is described in detail in the references [6, 8]. This general framework defined here could be applied to any multibody formulation.

For the tracking problem of $\widetilde{\mathcal{M}}$ we want to minimize the difference between multibody outputs and the prescribed reference that we have to track. This can be written in terms

of a minimization problem as follows:

$$\min_{\widetilde{\boldsymbol{x}},\widetilde{\boldsymbol{\lambda}},\widetilde{\boldsymbol{u}},\widetilde{\boldsymbol{y}}} \int_{T_0}^{T} ||\widetilde{\boldsymbol{y}}(t) - \boldsymbol{y}^*(t)||_{\boldsymbol{S}_y} \, \mathrm{d}t. \tag{30}$$

The quantities representing a set of multibody outputs is defined as

$$\widetilde{\boldsymbol{y}} = \widetilde{\boldsymbol{h}}(\widetilde{\boldsymbol{x}}), \tag{31}$$

The prescribed reference that should be tracked with the minimum error is defined by $\boldsymbol{y}^*$. The error is measured in the norm $||\bullet||_{\boldsymbol{S}_y} = (\bullet) \cdot \boldsymbol{S}_y^{\text{track}}(\bullet)$ with scaling matrix $\boldsymbol{S}_y^{\text{track}}$.

By the equations (32) and (33) we can express the initial conditions on the states, and the equality and inequality constraints on the inputs and outputs, respectively.

$$\widetilde{\boldsymbol{x}}(T_0) = \widetilde{\boldsymbol{x}}_0, \tag{32}$$

$$\boldsymbol{g}^{\text{track}}(\widetilde{\boldsymbol{y}}(t), \widetilde{\boldsymbol{u}}(t)) \in [\boldsymbol{g}(t)_{\text{min}}^{\text{track}}, \boldsymbol{g}(t)_{\text{max}}^{\text{track}}], \quad \forall\, t \in [T_0, T]. \tag{33}$$

These constraints can be augmented or changed to model some effects based on the conditions and requirements of the operation which can be limited control authority, performance envelope protection of the vehicle, the presence of obstacles, emergency conditions, etc.

The physical meaning of the multibody outputs $\widetilde{\boldsymbol{y}}$ is problem dependent. Typically, outputs of the flight mechanics model represent global vehicle states which defines its overall gross motion in terms of orientation, position, linear and angular velocities. Although multibody model states are more than flight mechanics states, it is always possible to compute some quantities that have the same physical meaning of the flight mechanics states. This can be handled by mapping corresponding states one to another. Also in the aspect of controls, those two models might have a different physical meaning. This problem will be discussed in detail in the section 3.

If we try to solve the problem given by equation (30) for a "perfect tracker case" we expect a tracking error $\widetilde{\boldsymbol{y}}(t) - \boldsymbol{y}^*(t)$ of zero. However, for this section we are assuming that the reference trajectory is given to us. This does not guarantee that the trajectory given

for the tracking problem is compatible with $\widetilde{\mathcal{M}}$, especially when the vehicle is forcing the limits of the performance envelope. Basically, it is difficult to generate feasible reference trajectories, so $\boldsymbol{y}^*$ might not be a possible solution of (29) and (31) subjected to (32) and (33). Computing a quasi-feasible $\boldsymbol{y}^*$ will be discussed later.

In order to make the solution of the equation (30) computationally feasible, we approximate the solution of the tracking problem using a non linear model predictive controller (NMPC).

4.1 Reduced Model

By the use of the reduced model we can have reasonable computational costs in solving the optimal control problem. This non linear reduced model $\mathcal{M}$ with the set of states, controls and parameters is represented by $\boldsymbol{y}$, $\boldsymbol{u}$, $\boldsymbol{p}$ respectively.

For this study, the reduced model parameters are based on the Neural Network parameters (weights and biases) which are optimized in order to satisfy a proper matching between reduced model outputs and full model outputs ($\boldsymbol{y} \approx \widetilde{\boldsymbol{y}}$). Clearly, both of them should be subjected to the same inputs.

We can define a *reference model* based on a mathematical model with the following expression.

$$\boldsymbol{f}_{\text{ref}}(\dot{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{u}) = 0. \tag{34}$$

Since we are working with two different level of detailed models, we have to be careful on the physical meanings of the controls. Controls of multibody model ($\widetilde{\boldsymbol{u}}$) might have different correspondance on the flight mechanics controls ($\boldsymbol{u}$). Detailed multibody model may include hydraulic actuators connected to the swash plate while reduced model controls include the rotor collective, longitudinal and lateral cyclics. Although these set of controls are clearly different they will change the pitch settings of the rotor blades at any given instant of time [1]. It is always possible to map one set of controls to another by the following equations.

$$\widetilde{\boldsymbol{u}} = \boldsymbol{m}(\boldsymbol{u}) \tag{35a}$$

$$\boldsymbol{u} = \boldsymbol{m}^{-1}(\widetilde{\boldsymbol{u}}) \tag{35b}$$

To make it simple we can consider $\widetilde{\boldsymbol{u}} = \boldsymbol{u}$ in the following.

A *reference model augmented by a neural network* can be written with the governing equation as

$$\boldsymbol{f}_{\text{ref}}(\dot{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{u}) = \boldsymbol{d}(\dot{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{u}), \tag{36}$$

where $\boldsymbol{d}(\dot{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{u})$ stands for the reference model error. The error is the defect of eq (34) encountered in matching the states of two different models (reduced and full, $\widetilde{\boldsymbol{y}} = \boldsymbol{y}$).

Now, we need to capture this defect $\boldsymbol{d}$ by using a single hidden layer neural network in the following manner where $\boldsymbol{W}$, $\boldsymbol{V}_{\dot{y}}$,$\boldsymbol{V}_y$ and $\boldsymbol{V}_u$ are matrices of weights and biases of the NN structure.

$$\boldsymbol{d}(\dot{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{u}) = \boldsymbol{W}^T \boldsymbol{\sigma}(\boldsymbol{V}_{\dot{\boldsymbol{y}}}^T \dot{\boldsymbol{y}} + \boldsymbol{V}_y^T \boldsymbol{y} + \boldsymbol{V}_u^T \boldsymbol{u}) + \boldsymbol{\varepsilon}, \tag{37}$$

Symbol σ used in eq (37) is an activation function which we can write as a vector form. Here, each element refers to corresponding hidden neuron.

$$\boldsymbol{\sigma}(\boldsymbol{\phi}) = (\sigma(\phi_1), \ldots, \sigma(\phi_{N_h}))^T \tag{38}$$

This non-linear activation function is chosen as the sigmoid function for the hidden layer where there are N_h number of hidden neurons. Also $\boldsymbol{\varepsilon}$ stands for functional reconstruction error which can be bounded as $||\boldsymbol{\varepsilon}||_2 \leq C_\varepsilon$, $C_\varepsilon > 0$ for some appropriately large number of hidden layer neurons [12].

As we mentioned before, parameters of the reduced model is defined as the synaptic weights and biases of the network.

$$\boldsymbol{p} = (\ldots, W_{ij}, \ldots, V_{\dot{y},ij}, \ldots, V_{y,ij}, \ldots, V_{u,ij}, \ldots)^T. \tag{39}$$

The neural network is responsible for minimizing the error between the network output and the desired output. To ensure this, the network is subjected to the training procedure with an effective learning algorithm, which is a gradient descent based error-correction algorithm. Since we have the information on the multibody model (plant) from previous steering, we can define the error between two models in the following way,

$$network \;\; output = \boldsymbol{W}^T \boldsymbol{\sigma}(\boldsymbol{V}_{\dot{y}}^T \dot{\widetilde{\boldsymbol{y}}}^* + \boldsymbol{V}_y^T \widetilde{\boldsymbol{y}}^* + \boldsymbol{V}_u^T \boldsymbol{u}^*) \tag{40a}$$

$$desired \;\; output = \boldsymbol{f}_{\text{ref}}(\dot{\widetilde{\boldsymbol{y}}}^*, \widetilde{\boldsymbol{y}}^*, \boldsymbol{u}^*) \tag{40b}$$

$$E = ||\boldsymbol{W}^T \boldsymbol{\sigma}(\boldsymbol{V}_{\dot{y}}^T \dot{\widetilde{\boldsymbol{y}}}^* + \boldsymbol{V}_y^T \widetilde{\boldsymbol{y}}^* + \boldsymbol{V}_u^T \boldsymbol{u}^*) - \boldsymbol{f}_{\text{ref}}(\dot{\widetilde{\boldsymbol{y}}}^*, \widetilde{\boldsymbol{y}}^*, \boldsymbol{u}^*)||_2. \tag{40c}$$

where $\widetilde{\boldsymbol{y}}^*$ is the plant outputs and $\boldsymbol{u}^*$ is the given control inputs.

Several methods are available in order to solve this optimization problem [13, 14].

By approximating the defect of the reference model with the neural network, we are able to capture the complete output behavior and matching $\boldsymbol{y} \approx \widetilde{\boldsymbol{y}}$. The success of this approach is problem independent.

We can express the reduced model in a compact notation, which will be used during the rest of the study.

$$\boldsymbol{f}(\dot{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{p}) = 0. \tag{41}$$

There can be two possibilities for parameters. Parameter $\boldsymbol{p}$ can be independent or dependent of time. If $\boldsymbol{p}$ is independent of time, then it can define the reduced model through the whole maneuver. If $\boldsymbol{p}$ is varying with time ($\boldsymbol{p} = \boldsymbol{p}(t)$), then the reduced model parameters are based on local nature.

4.2 Model Predictive Tracking

As a main concept, we use the reduced model $\mathcal{M}$ in predicting the future behavior of the plant $\widetilde{\mathcal{M}}$ steered by the control inputs. The principle of the model predictive tracking can be seen in Figure (22).

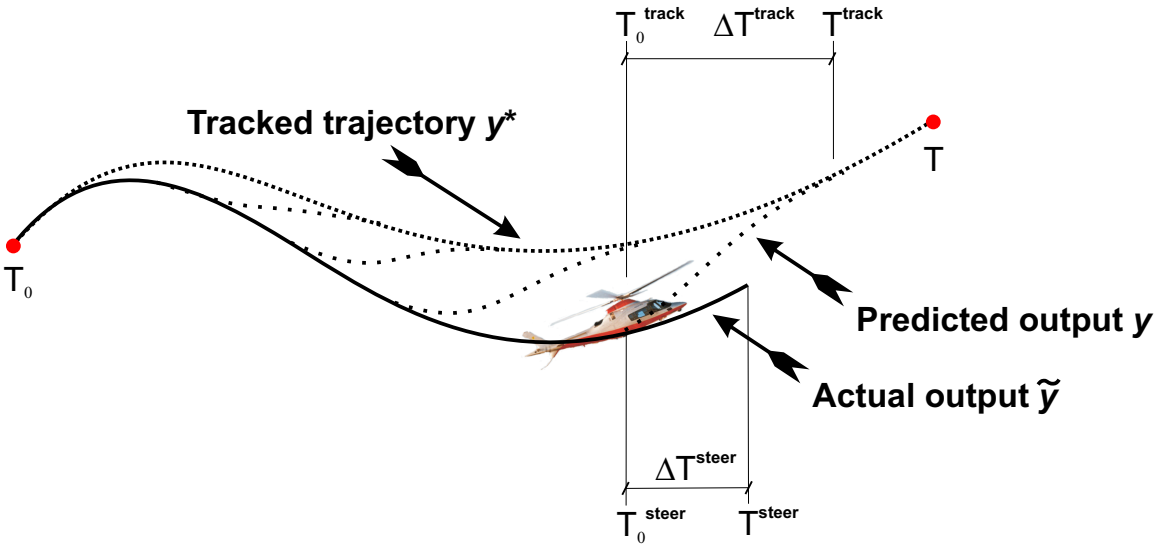

Figure 22: Model predictive control.

On a finite horizon (tracking window), we solve the open-loop optimal control problem with a cost function for the reduced model. Tracking cost function is based on the tracking error and some weighted control activity shown in equation (43). Also the optimizer ensures the satisfaction of some input and output constraints. As a solution of this open-loop model predictive tracking problem, we get some computed control actions that we can use to steer the multibody model (plant) $\widetilde{\mathcal{M}}$ only a short time horizon (steering window). This steering window should be short enough to capture the short period modes of the plant for better performance.

As expected after steering the plant $\widetilde{\mathcal{M}}$, due to the mismatch between reduced model and plant, the actual outputs will be different from the predicted ones. As a next step, the tracking problem is solved again in a shifted horizon starting from the point where the steering of plant is ended. This procedure is iteratively carried out until we reach the end of the maneuver. By the receding horizon approach we apply the feedback to the problem.

Assuming that the reference outputs ($\boldsymbol{y}^*$) are given or achieved by solving the trajectory optimization problem for the whole maneuver by using the reduced model, we can go through and explain the tracking and steering procedure by a mathematical formulation. Also we can use some current estimation for model parameters ($\boldsymbol{p}^*$) which are neural network weights and biases.

Beginning of the tracking and steering windows are defined as $t = T_0^{\text{track}} = T_0^{\text{steer}}$.

Also end of the tracking window is defined with the window size ΔT^{track}, $T^{\text{track}} = T_0^{\text{track}} + \Delta T^{\text{track}}$. Given initial conditions on the plant states are $\widetilde{\boldsymbol{x}}(T_0^{\text{track}}) = \widetilde{\boldsymbol{x}}_0$. Output initials conditions are defined as $\widetilde{\boldsymbol{y}}_0 = \widetilde{\boldsymbol{h}}(\widetilde{\boldsymbol{x}}_0)$. The model predictive tracking problem is defined with the following equations.

$$\min_{\boldsymbol{y},\boldsymbol{u}} J^{\text{track}}, \tag{42a}$$

$$\text{with: } J^{\text{track}} = \int_{T_0^{\text{track}}}^{T^{\text{track}}} M(\boldsymbol{y}, \boldsymbol{y}^*, \boldsymbol{u})\, \mathrm{d}t, \tag{42b}$$

$$\text{s.t.: } \boldsymbol{f}(\dot{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{p}^*) = 0, \tag{42c}$$

$$\boldsymbol{g}^{\text{track}}(\boldsymbol{y}, \boldsymbol{u}) \in [\boldsymbol{g}_{\min}^{\text{track}}, \boldsymbol{g}_{\max}^{\text{track}}], \tag{42d}$$

$$\boldsymbol{y}(T_0^{\text{track}}) = \widetilde{\boldsymbol{y}}_0. \tag{42e}$$

The tracking cost is defined by the following equation (43). The first term stands for the tracking error, while the second and the third terms are the quadratic terms based on control actions and control rates which are used to get smooth control behaviors by changing the weighting matrices.

$$M(\boldsymbol{y}, \boldsymbol{y}^*, \boldsymbol{u}) = ||\boldsymbol{y} - \boldsymbol{y}^*||_{\boldsymbol{S}_y^{\text{track}}} + ||\boldsymbol{u}||_{\boldsymbol{S}_u^{\text{track}}} + ||\dot{\boldsymbol{u}}||_{\boldsymbol{S}_{\dot{u}}^{\text{track}}} \tag{43}$$

For the plant steering formulation, we are using the controls $\boldsymbol{u}^*(t)$ that we have found already with the solution of the problem (42). In this case, we define $t \in \Omega^{\text{steer}} = (T_0^{\text{steer}}, T^{\text{steer}})$ where $T^{\text{steer}} = T_0^{\text{steer}} + \Delta T^{\text{steer}}$ is the end of the steering window with size ΔT^{steer}. By the given controls $\boldsymbol{u}^*$, the plant $\widetilde{\mathcal{M}}$ is subjected to the following standard initial value problem :

$$\widetilde{\boldsymbol{f}}(\dot{\widetilde{\boldsymbol{x}}}, \widetilde{\boldsymbol{x}}, \widetilde{\boldsymbol{\lambda}}, \boldsymbol{u}^*) = 0, \tag{44a}$$

$$\widetilde{\boldsymbol{c}}(\dot{\widetilde{\boldsymbol{x}}}, \widetilde{\boldsymbol{x}}) = 0, \tag{44b}$$

$$\widetilde{\boldsymbol{x}}(T_0^{\text{steer}}) = \widetilde{\boldsymbol{x}}_0, \tag{44c}$$

This problem provides a solution in terms of $\widetilde{\boldsymbol{x}}(t)$ and $\widetilde{\boldsymbol{\lambda}}(t)$ for $t \in \Omega^{\text{steer}}$. The solution that we get at the end of the steering window provides the new initial condition for the next step.

Some important characteristic details of the model predictive control approach for the solution of tracking problem are listed below:

- Future behavior prediction of the plant is based on the finite horizon, rather than the infinite horizon (T_∞), since shorter horizons require small computational costs. However, longer horizons will provide improved stability and performance.

- By the computed controls from the optimizer, outputs of the system $\widetilde{\mathcal{M}}$ will shift away from the predicted solutions. When we keep the steering window (ΔT^{steer}) longer, this shift will be greater. However, longer windows will decrease the computational cost by decreasing the number of model predictive tracking problems that need to be solved.

- The modeling errors, causing a mismatch between the predicted and actual outputs, can be augmented by the adaption of the neural network parameters (model parameters) with on-line training after each steering action is done for the corresponding window.

- Using a non-linear model of the plant improves the performance in a way of matching the plant dynamics.

- Trade-off's are possible for the first and second comments in a sense of window length of tracking and steering phase.

4.3 Numerical Solution of the Model Predictive Tracking and Steering Problems

For the solution of the model predictive tracking problem there can be two possible strategies. First option is the indirect approach which is based on deriving the optimal control equations i.e state, adjoint (co-state), control equations. These set of equations define an infinite dimensional non-linear multi-point boundary value problem. Second option is the direct approach, where one first discretizes the optimal control equations and renders the problem finite dimensional by using a suitable discretization methodology and then optimizes the problem.

There are some disadvantages of the indirect approach compared with the direct approach. These are mentioned below.

- Indirect approach requires the derivation of optimal control equations which can be a complicated and tedious task for comprehensive systems or a rotorcraft. Also it is not a flexible approach, since each time a new problem is posed and a new derivation of the relevant derivatives is called.

- Indirect method resides in the necessity of providing suitable initial guesses for all variables including states of the system and co-states (adjoint variables). The adjoint variables are not physical quantities and is very nonintuitive so it is a difficult task to initialize those values. Even with a reasonable guess the numerical solution of the co-state equations will be ill-conditioned. This makes the method non-robust.

- In the indirect approach we need the constrained and unconstrained sub-arcs as a priori for problems with state inequalities. This is quite difficult since if we don't know the number of constrained sub-ars, the number of iteration variables is also unknown. Furthermore, the sequence of arcs is unknown which makes it difficult to impose the correct junction conditions and define the arc boundaries. This reduces the generality of the method.

Based on this information, we use the direct transcription methodology for the numerical solution of the model predictive tracking problem (optimal control problem in a tracking

window). The reduced model governing equations are discretized on a computational grid of the tracking window. An appropriate numerical method is used for this process. As a result of the discretization, we define a set of discrete state and control parameters on this computational grid. Also we express the cost function and constraints in terms of the discrete parameters. The problem turns to a non-linear programming problem. Numerical solution of this non-linear discrete parameter optimization problem approximates its infinite dimensional correspondent problem (42).

In order to describe the numerical solution more precisely, let us consider a temporal domain as $\Omega^{\text{track}} = (T_0^{\text{track}}, T^{\text{track}})$. This is temporal domain is partitioned as follows:

$$T_0^{\text{track}} \equiv t_0 < t_1 < \ldots < t_{n-1} < t_n \equiv T^{\text{track}}, \tag{45}$$

where $n \geq 1$, and $t_{i+1} = t_i + h^i$, $i = 0, \ldots, n-1$. $\mathcal{T}_h^{\text{track}}$ represents the grid, made of n elements, associated with the partition. K represents a generic element. $T^i = [t_i, t_{i+1}]$ is the time interval spanned by each element. Each element has a left vertex ∂K^L contributed with time t_i and right vertex ∂K^R contributed with time t_{i+1}. We can write the size of each element as $h^i = h = (T^{\text{track}} - T_0^{\text{track}})/n$, since they are held constant throughout the grid.

The infinite dimensional unknown fields $\boldsymbol{y}(t)$, $\boldsymbol{u}(t)$ are approximated with functions $\boldsymbol{y}_h$, $\boldsymbol{u}_h$ chosen within the finite dimensional spaces by the use of the finite element method. The restriction of these approximations to generic element K, is noted by $\boldsymbol{y}_h|_K$ and $\boldsymbol{u}_h|_K$. The following equation shows that the state approximations evaluated on the right vertex of an element K_i are equal to the state approximations evaluated on the left vertex of the neighboring element K_{i+1}.

$$\boldsymbol{y}_h|_{\partial K_i^R} = \boldsymbol{y}_h|_{\partial K_{i+1}^L} \tag{46}$$

Continuity of the states at the element interfaces provides the initial conditions on each element as the value of the final states on the preceding element. However, there is no initial conditions on the controls so that the control approximations $\boldsymbol{u}_h$ should be discontinuous across element interfaces.

The functions $\boldsymbol{y}_h$ and $\boldsymbol{u}_h$ can be defined in terms of some discrete parameters $\boldsymbol{y}_d$, $\boldsymbol{u}_d$ such that $\boldsymbol{y}_h = \boldsymbol{y}_h(\boldsymbol{y}_d)$ and $\boldsymbol{u}_h = \boldsymbol{u}_h(\boldsymbol{u}_d)$ on the computational grid $\mathcal{T}_h^{\text{track}}$. Generally, the vector of discrete parameters $\boldsymbol{y}_d$ will contain the state unknowns at the grid nodes, $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n$. Also there will be additional n_s internal unknowns (stages) per time step $\boldsymbol{y}_i^1, \boldsymbol{y}_i^2, \ldots, \boldsymbol{y}_i^{n_s}$, for some schemes such as Runge-Kutta and finite element methods. The vector form of discrete state parameters can be written in a general form as

$$\boldsymbol{y}_d = (\ldots, \boldsymbol{y}_i, \boldsymbol{y}_i^1, \boldsymbol{y}_i^2, \ldots, \boldsymbol{y}_i^{n_s}, \boldsymbol{y}_{i+1}, \ldots)^T. \tag{47}$$

Furthermore, the vector of the discrete control parameters $\boldsymbol{u}_d$ can be written in a similar way depending on the numerical method.

The discretized formulation of the model predictive tracking problem can be written as

$$\min_{\boldsymbol{y}_d, \boldsymbol{u}_d} J_h^{\text{track}}, \tag{48a}$$

$$\text{with: } J_h^{\text{track}} = \int_{T_0^{\text{track}}}^{T^{\text{track}}} M(\boldsymbol{y}_h, \boldsymbol{y}_h^*, \boldsymbol{u}_h)\, \mathrm{d}t, \tag{48b}$$

$$\text{s.t.: } \boldsymbol{f}_h(\boldsymbol{y}_h|_K, \boldsymbol{u}_h|_K, \boldsymbol{p}^*) = 0 \quad \forall K \in \mathcal{T}_h^{\text{track}}, \tag{48c}$$

$$\boldsymbol{g}_h^{\text{track}}(\boldsymbol{y}_h|_K, \boldsymbol{u}_h|_K) \in [\boldsymbol{g}_{\min,K}^{\text{track}}, \boldsymbol{g}_{\max,K}^{\text{track}}] \quad \forall K \in \mathcal{T}_h^{\text{track}}, \tag{48d}$$

$$\boldsymbol{y}_h(T_0^{\text{track}}) = \widetilde{\boldsymbol{y}}_0. \tag{48e}$$

In the equation (48), a discretized version of J^{track} as given in (42b) is represented by (48b) where the integral being evaluated with some appropriate quadratic rule. Equation (48c) represents a discretized version of reduced model governing equation (41) on each element K of the computational grid. These equations are coupled from the conditions mentioned in equation (46). Finally, another set of constraints represented by (48d) which is a discretized version of input and output constraints of problem (42). Unknowns of the optimization problem are the set of discrete state and control parameters $\boldsymbol{y}_d$, $\boldsymbol{u}_d$.

The transcription process is based on the finite element in time formulation of explained detailed in Reference [17].

Second phase of the problem is based on solving the initial value problem with known control actions to steer the plant. $\widetilde{\mathcal{T}}_h^{\text{steer}}$ is the grid used for advancing in time with the multibody model in Ω^{steer}. In general, typical time step size in $\widetilde{\mathcal{T}}_h^{\text{steer}}$ is smaller than the typical time step size in $\mathcal{T}_h^{\text{track}}$. Much finer solution scales need to be resolved in steering case. There is a need to map the controls $\boldsymbol{u}_h^*$ obtained on $\mathcal{T}_h^{\text{track}}$ from the solution of problem (48) onto the multibody steering grid $\mathcal{T}_h^{\text{steer}}$, since numerical method used for integrating the multibody model equations can be different form the numerical method used for discretizing the optimal control problem. This mapping is based on the grids and numerical methods used at the tracking and steering levels which can be simply indicated as follows

$$\boldsymbol{u}_h^*|_{\widetilde{\mathcal{T}}_h^{\text{steer}}} = \mathcal{P}(\boldsymbol{u}_h^*|_{\mathcal{T}_h^{\text{track}}}), \tag{49}$$

where $\mathcal{P}(\bullet)$ is an appropriate mapping operator.

The discretized version of problem (44) can be written as

$$\widetilde{\boldsymbol{f}}_h(\widetilde{\boldsymbol{x}}_h|_K, \widetilde{\boldsymbol{\lambda}}_h|_K, \boldsymbol{u}_h^*|_K) = 0 \quad \forall K \in \widetilde{\mathcal{T}}_h^{\text{steer}}, \tag{50a}$$

$$\widetilde{\boldsymbol{c}}_h(\widetilde{\boldsymbol{x}}_h|_K) = 0 \quad \forall K \in \widetilde{\mathcal{T}}_h^{\text{steer}}, \tag{50b}$$

$$\widetilde{\boldsymbol{x}}_h(T_0^{\text{steer}}) = \widetilde{\boldsymbol{x}}_0. \tag{50c}$$

The discrete equations in (50a) and (50b) are solved on each element sequentially. Initial conditions ar provided as in (50c) for the first element. For all subsequent elements they are given by the conditions in equation (46). As a result the outputs are obtained as

$$\widetilde{\boldsymbol{y}}_h|_K = \widetilde{\boldsymbol{h}}(\widetilde{\boldsymbol{x}}_h|_K), \quad \forall K \in \widetilde{\mathcal{T}}_h^{\text{steer}}. \tag{51}$$

The numerical integration of the multibody dynamics equations is based on the methods described in Reference [8] and references therein.

4.4 Model Adaption

In order to increase the reliability of the reduced model, we should get a reasonable match between the predicted and actual outputs. We can minimize this mismatch subjecting the reduced model and plant to the same control inputs and using a *local adaption* method to augment the neural network based model parameters, iteratively.

Assuming $\boldsymbol{u}_h^*$ some given control inputs and $\widetilde{\boldsymbol{y}}_h^*$ the resulting multibody outputs, the model adaption problem can be formulated as :

$$J_h^{\text{adapt}} = \int_{T_0^{\text{adapt}}}^{T^{\text{adapt}}} E \mathrm{d}t. \tag{52}$$

For model augmentation with the neural network, the definition of E is given in equation (40c) where it represents the defect of the reference model.

In the local adaptation problem initial and final times are chosen to be $T_0^{\text{adapt}} = T_0^{\text{steer}}$ and $T^{\text{adapt}} = T^{\text{steer}}$ respectively.

The neural network augmented reference model are allowed to change based on the parameters throughout the maneuver. The idea is totally based on using the local information that we get from the previous steering window to correct the estimate parameters ($\boldsymbol{p} = \boldsymbol{p}(t)$).

In the following formulation we can see that current parameters $\boldsymbol{p}_{\text{curr}}$ are adjusted to the new parameters $\boldsymbol{p}_{\text{new}}$.

$$\boldsymbol{p}_{\text{new}} = \boldsymbol{p}_{\text{curr}} + \Delta\boldsymbol{p}. \tag{53}$$

Parameter correction $\Delta\boldsymbol{p}$ can be found by the backpropagation algorithm [14] using the steepest descent methodology with a learning rate of η.

$$\Delta\boldsymbol{p} = -\,\eta \frac{\partial J_h^{\text{adapt}}}{\partial \boldsymbol{p}}\bigg|_{\boldsymbol{p}_{\text{curr}}} \tag{54}$$

4.5 Planning of Multibody Trajectories

As we mentioned before, generating feasible tracking trajectories for the multibody models is a difficult problem. We have to ensure that the generated trajectory is compatible with the vehicle dynamics. So far we assumed that this reference trajectory is given as $\boldsymbol{y}^*$.

In order to plan a trajectory, we have to clearly define the problem and construct an optimized cost function as a vehicle performance index with some additional constraints on states and controls which are based on the maneuver requirements such as minimum time, minimum power, etc...

If we try to solve the optimal control problem using the multibody model, we will not achieve a feasible computational cost so the reduced model is used for this phase. As a solution of this problem, we get controls, states and possibly the final time (T), since we assumed a free final time problem. In order to guarantee that the resulting trajectory is trackable, $\mathcal{M}$ should represent the $\widetilde{\mathcal{M}}$ with sufficient accuracy.

Having some initialized neural network weights and biases represented as model parameters $\boldsymbol{p}^*$, we can construct the optimal control problem as follows :

$$\min_{\boldsymbol{y},\boldsymbol{u},T} J^{\text{plan}}, \tag{55a}$$

$$\text{with: } J^{\text{plan}} = \phi(\boldsymbol{y},\boldsymbol{u})\big|_T + \int_{T_0}^{T} L(\boldsymbol{y},\boldsymbol{u})\,\mathrm{d}t, \tag{55b}$$

$$\text{s.t.: } \boldsymbol{f}(\dot{\boldsymbol{y}},\boldsymbol{y},\boldsymbol{u},\boldsymbol{p}^*) = 0, \tag{55c}$$

$$\boldsymbol{g}^{\text{plan}}(\boldsymbol{y},\boldsymbol{u},T) \in [\boldsymbol{g}^{\text{plan}}_{\min}, \boldsymbol{g}^{\text{plan}}_{\max}], \tag{55d}$$

$$\boldsymbol{\psi}(\boldsymbol{y}(T_0)) \in [\boldsymbol{\psi}_{0_{\min}}, \boldsymbol{\psi}_{0_{\max}}], \tag{55e}$$

$$\boldsymbol{\psi}(\boldsymbol{y}(T)) \in [\boldsymbol{\psi}_{T_{\min}}, \boldsymbol{\psi}_{T_{\max}}]. \tag{55f}$$

The constraint equations (55d) represent point and integral constraints and bounds on states and controls, while (55e) and (55f) are initial and final boundary conditions on the states respectively.

Methods for the optimal control problem are currently based on the discretization process which makes the problem finite dimensional. Therefore, instead of first deriving the equations and discretizing them, the reverse order is much more convenient to solve the problem, where one discretizes the equations of equilibrium, constraints and the cost function in order to make the problem finite dimensional. Then this discrete problem is solved. This is called *direct approach* or *non-linear programming (NLP)* problem [15]. The discretized problem is given below.

$$\min_{\boldsymbol{y}_d, \boldsymbol{u}_d, T} J_h^{\text{plan}} \tag{56a}$$

$$\text{with: } J_h^{\text{plan}} = \phi(\boldsymbol{y}_h, \boldsymbol{u}_h)\big|_T + \int_{T_0}^{T} L(\boldsymbol{y}_h, \boldsymbol{u}_h)\,\mathrm{d}t \tag{56b}$$

$$\text{s.t.: } \boldsymbol{f}_h(\boldsymbol{y}_h|_K, \boldsymbol{u}_h|_K, \boldsymbol{p}^*) = 0 \quad \forall K \in \mathcal{T}_h^{\text{plan}} \tag{56c}$$

$$\boldsymbol{g}_h^{\text{plan}}(\boldsymbol{y}_h|_K, \boldsymbol{u}_h|_K, T) \in [\boldsymbol{g}_{\text{min},K}^{\text{plan}}, \boldsymbol{g}_{\text{max},K}^{\text{plan}}] \quad \forall K \in \mathcal{T}_h^{\text{plan}} \tag{56d}$$

$$\boldsymbol{\psi}(\boldsymbol{y}_h(T_0)) \in [\boldsymbol{\psi}_{0_{\min}}, \boldsymbol{\psi}_{0_{\max}}] \tag{56e}$$

$$\boldsymbol{\psi}(\boldsymbol{y}_h(T)) \in [\boldsymbol{\psi}_{T_{\min}}, \boldsymbol{\psi}_{T_{\max}}] \tag{56f}$$

First a crude grid is used with rough initial guesses. Then the solution of this NLP problem is projected to a finer grid where the previous solution is used as an initial guess. This yields to a reference trajectory $\boldsymbol{y}_h^*$ on the planning problem grid $\mathcal{T}_h^{\text{plan}}$.

4.6 Integrated Adaptive Planning and Tracking: the Multi-Model Steering Algorithm

So far, we defined the methodology to get the reference trajectory and receding horizon procedure to steer the plant $\widetilde{\mathcal{M}}$. Also we talked about the adaption procedure for the reduced model in order to minimize the tracking errors based on the mismatch of these two models in previous sections. Now we have to address the possible way to improve this tracking performance.

We can alleviate the tracking performance by proposing an integrated adaptive planning and tracking algorithm that tries to increase the compatibility between tracking trajectory,

reference model and plant. This methodology is used in a similar application named as multi-model steering algorithm (MMSA) in reference [2]. The idea is to iterate between planning, tracking and adaption phases until we reach the desired error between the planned and the realized trajectories. Error range should be small enough in order to ensure compatibility. Also, if we don't recognize any further model improvement we should stop the iterations.

The algorithm is discussed for overall procedure based on local adaption. We can go through all the steps in order to make the discussion more clear. In order to reflect the local characteristics of the solution, instead of using a single set of parameters as $\boldsymbol{p}$ we introduced a set of reduced model parameters as $\boldsymbol{p}_i$ (i defines the parameters for the corresponding window during the maneuver).

As a first step, the trajectory planning problem (56) is solved with some randomly selected initialized values of $\boldsymbol{p}_i$ (small enough) which are neural network synaptic weights and biases (used as model parameters). We got an estimated tracking trajectory $\boldsymbol{y}_h^*$. After that the multibody model (plant) is steered using model predictive tracking. During this process the model predictive tracking problem is solved in order to get some controls. Since these controls are on the tracking grid they are projected to the multibody (steering) grid. Then the steering problem is solved.

Maneuver can be repeated several times. When the maneuver is repeated the sequence of neural networks progressively learn how to correct the local prediction errors. An upper bound can be set to stop the repetition of the maneuver. It can also be stopped either the convergence of the tracking trajectory has been almost achieved or when it is recognized that the neural network output doesn't change much with the same inputs. The last realized trajectory of the multibody model is reconstructed and accordingly the tracking error is evaluated. In order to reduce this error a new planning / multiple-tracking-steering iteration is initiated.

In order to solve problem (56) again, we need a single constant value of the model parameters. However a sequence of parameters has been obtained with local adaption. For this reason we need to generate a single set of parameters (global parameters) starting from

the sequence of local parameters $\boldsymbol{p}_i$. One way of achieving this is to average the neural network synaptic weights and biases. This averaged value is taken as a single set of global parameters $\boldsymbol{p}^*$. Accordingly replanning is carried out. Then multiple tracking and steering steps are continued until desired convergence is achieved.

CHAPTER V

NUMERICAL APPLICATIONS

The procedure of simulation of the multibody maneuvering has been proposed so far. Now, we would like to assess the performance of this procedure by an application based on a violent maneuvering of a rotorcraft.

The multibody model formulation is totally based on the comprehensive finite element methodology [6]. Equations of equilibrium are written in cartesian inertial frame and Lagrange multiplier technique is used in modeling the constraints. Numerical integration of DAEs' are performed with non-linearly unconditionally stable energy decaying scheme [8, 9].

Multibody model is a standard medium size multi engine helicopter with a mass around 9000kg and includes four bladed articulated rotor modelled with beam elements and revolute joints connected to a rigid fuselage. Effects such as unsteadiness, radial drag, tip losses, sweep and twist are considered by using lifting line formulation. Inflow correction is used for non-uniform inflow based on the theory given in reference [16].

Dynamic rotorcraft equations of the reference model are based on two dimensional longitudinal dynamics [3, 10]. Uniform inflow and blade element theory is used in computation of rotor forces and moments. Also quasi-steady flapping dynamics are considered in finding the attitude of the rotor. Furthermore, downwash angle at the tail due to the main rotor is included.

Reduced model states and controls are defined as follows :

$$\boldsymbol{y} = (X, Z, \Theta, V_X, V_Z, q, \omega) \tag{57a}$$

$$\boldsymbol{u} = (\theta_{0_{MR}}, \theta_{0_{TR}}, A_1, B_1, P) \tag{57b}$$

For states, X and Z (positive downward) represent the position vector of the vehicle

center of gravity. V_X and V_Z are their time rates. Θ (positive nose up) is the pitch angle while q is the pitch rate. Also ω represents the angular velocity of the rotor.

For controls, $\theta_{0_{MR}}$ is the main rotor collective, $\theta_{0_{TR}}$ is the tail rotor collective, A_1, B_1 are lateral and longitudinal cyclics respectively and P represents the power available.

Helicopter Obstacle Avoidance Problem

The maneuver illustrated in figure (23) involves violent maneuvers in a hostile environment. In this example, the helicopter will be in level flight in the proximity of the ground. Then in order to avoid from the obstacle the helicopter will be subjected to a violent pull up and immediately violent pull down. The helicopter should achieve its low steady flight condition back in minimum time.

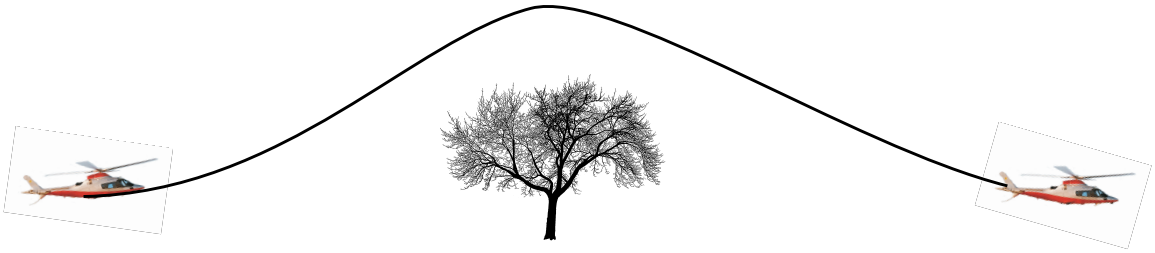

Figure 23: Helicopter obstacle avoidance problem.

The planning part of the problem is characterized by an unknown final time (free final time) T and an unknown internal event T_1, $T_1 \in [T_0, T]$ which represent the instant when the vehicle passes over the obstacle. The planning cost of the problem (55) can be expressed by the following formulation:

$$J^{\text{plan}} = T + \frac{1}{T - T_0} \int_{T_0}^{T} \left(w_B \, \dot{B}_1^2 + w_\theta \, \dot{\theta}_{0_{MR}}^2 + w_P \, \dot{P}^2 \right) \mathrm{d}t. \tag{58}$$

First term T enforces the minimum time condition that we need to achieve as a mission requirement. The terms in the integral penalize high cyclic, collective and power rates [1]. Tunable weighting factors are chosen as $w_B = 1000$, $w_\theta = 500$ and $w_P = 100$.

Throughout the maneuver we impose bounds on the collective and cyclic controls

$$\theta_{0_{MR}} \in [-5^o, 20^o], \quad \theta_{0_{TR}} \in [-10^o, 30^o] \tag{59a}$$

$$A_1 \in [-12^o, 12^o], \quad B_1 \in [-20^o, 20^o] \tag{59b}$$

and on their rates

$$\dot{\theta}_{0_{MR}}, \dot{\theta}_{0_{TR}}, \dot{A}_1, \dot{B}_1 \in [-16^o s^{-1}, 16^o s^{-1}] \tag{60}$$

on the rotor speed

$$\omega \in [-207, 207] rpm \tag{61}$$

on the unknown times

$$T_1 \in [1s, 30s], \quad T \in [1s, 20s] \tag{62}$$

on the power and the power rate

$$P \leq 2500hp, \quad \dot{P(t)} \leq 500hps^{-1} \tag{63}$$

and finally on the obstacle avoidance condition T_1

$$Z(T_1) \geq 60m. \tag{64}$$

The initial conditions for equation (55e) correspond to level flight trim states which are set as follows:

$$X(0) = Z(0) = 0\ m, \quad \Theta(0) = 0.26\ deg, \quad V_X(0) = 50\ ms^{-1}, \quad V_Z(0) = 0\ ms^{-1}, \tag{65}$$

$$q(0) = 0\ deg\, s^{-1} \quad \omega(0) = 207\ rpm.$$

Finally, the final conditions are expressed as :

$$V_X(T) = 50\ ms^{-1}, \quad V_Z(T) = 0\ ms^{-1}, \quad q(T) = 0\ deg\, s^{-1}. \tag{66}$$

The planning problem is solved by using a uniform grid of 40 time steps. Tracking and steering windows were selected as $\Delta T^{\text{track}} = 2.0\,s$ and $\Delta T^{\text{steer}} = 0.2s$ respectively. The activation frequency of the NMP controller and accordingly the frequency of the feedback information is based on the steering window length. This window should be small enough to capture the short period mode of the vehicle, which is approximately equal to $1\,s$.

Figure (24) illustrates the effect of the local adaption on the final compatibility between the planned trajectory and the effectively realized one. The procedure is ended after 14 planning / tacking-steering / adaption iterations since no improvement is realized after 14 iterations. Figure (25) and (26) show typical results of fuselage pitch and vehicle airspeed respectively before and after the iterations. Lines marked with Δ correspond to the solution of the planning problem while the solid lines correspond to the realized multibody model outputs. With respect to the initial solution before the adaption we can clearly see that the tracking error is decreased. The planned and the tracked results are close compared with the initial case. We can also see that the maneuver time is increasing when we move on the replanning phase due to the adjustment in the reduced model. Also the controls are getting smoother and bounded when we move on the procedure (27). As seen from the figures the tracking error is reasonably small. Also another case is studied by decreasing the initial speed of the helicopter to 30m/s. The figures are in the appendix A. Both cases showed that the NMPC strategy with neural network promises good solutions for the future work in this area.

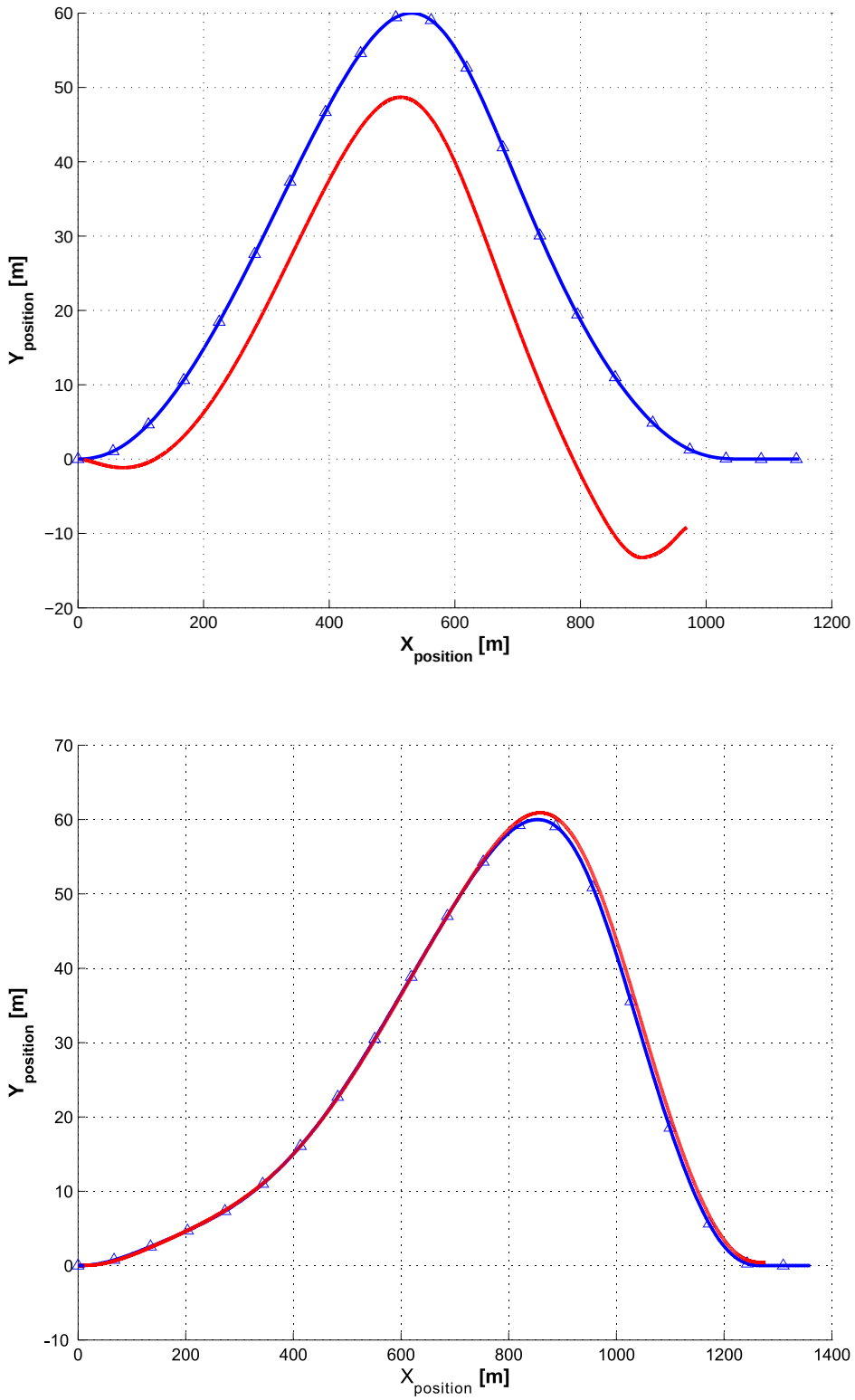

Figure 24: Helicopter obstacle avoidance maneuver with initial speed 50m/s. Trajectory flown by the reduced (Δ line) and multibody (solid line) models, before (top), after (bottom) fourteen planning / tracking steering / adaption iterations.

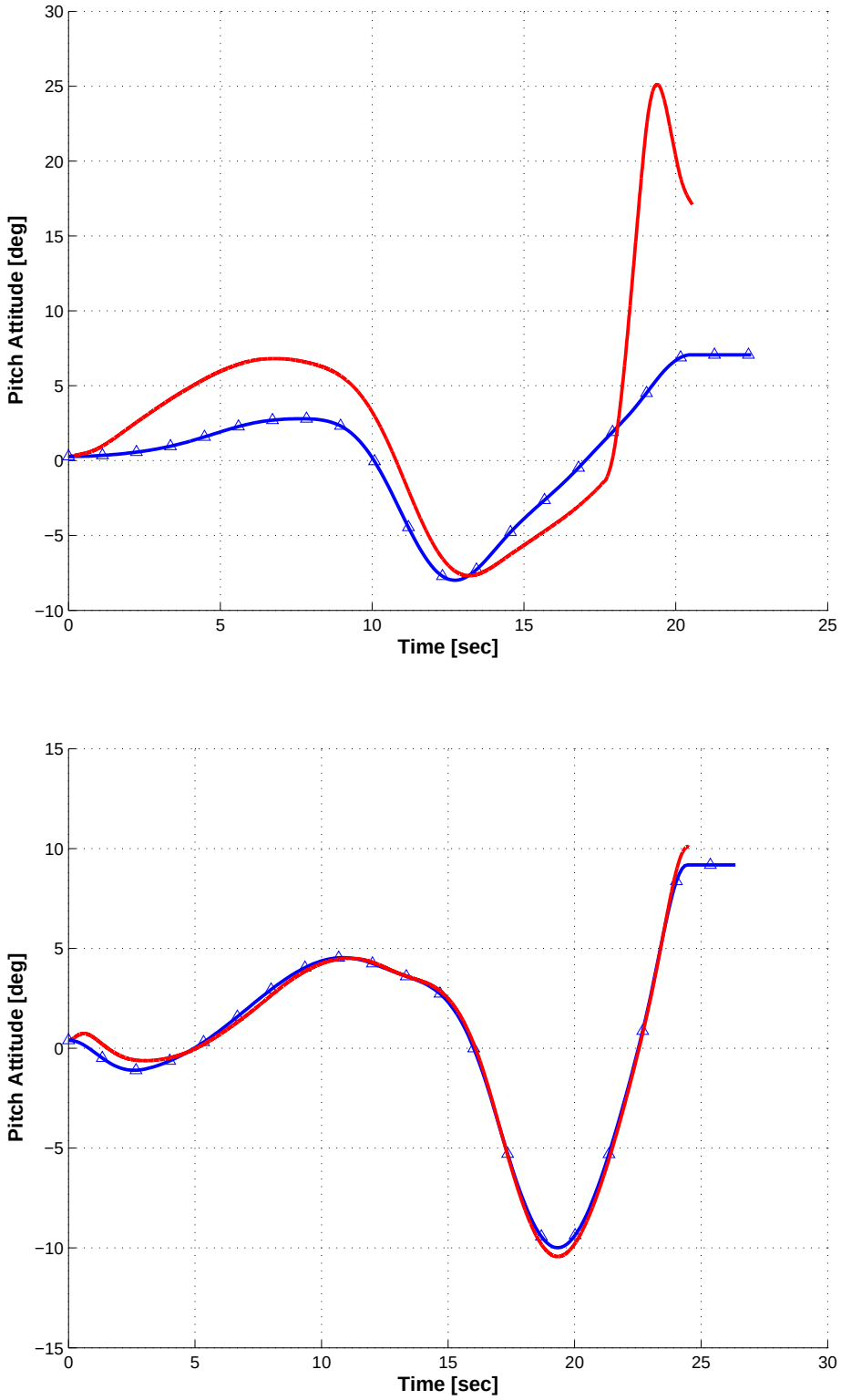

Figure 25: Helicopter obstacle avoidance maneuver with initial speed 50m/s. Fuselage pitch for the reduced (Δ line) and multibody (solid line) models, before (top), after (bottom) fourteen planning / tracking steering / adaption iterations.

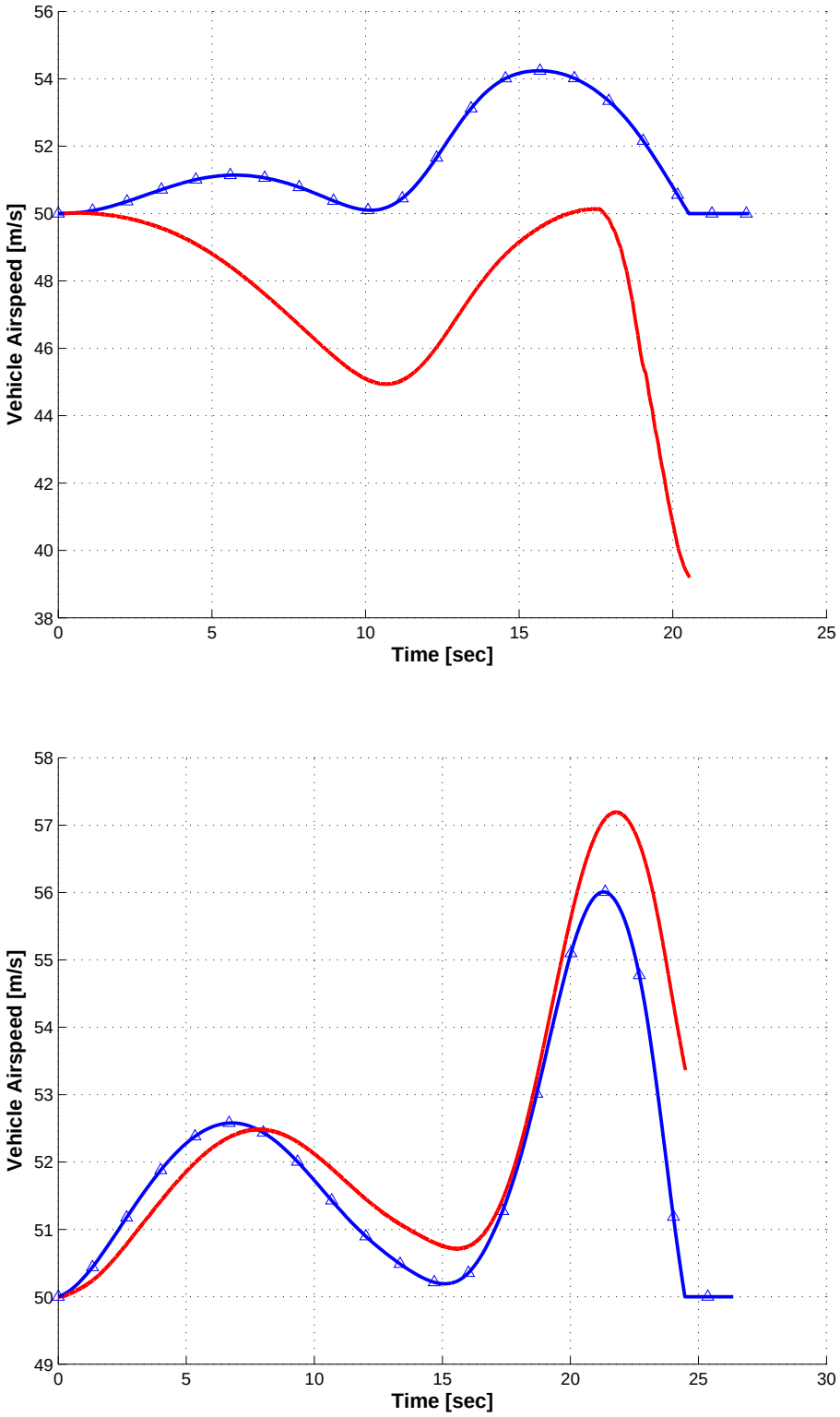

Figure 26: Helicopter obstacle avoidance maneuver with initial speed 50m/s. Rotorcraft speed for the reduced (Δ line) and multibody (solid line) models, before (top), after (bottom) four planning / tracking steering / adaption iterations.

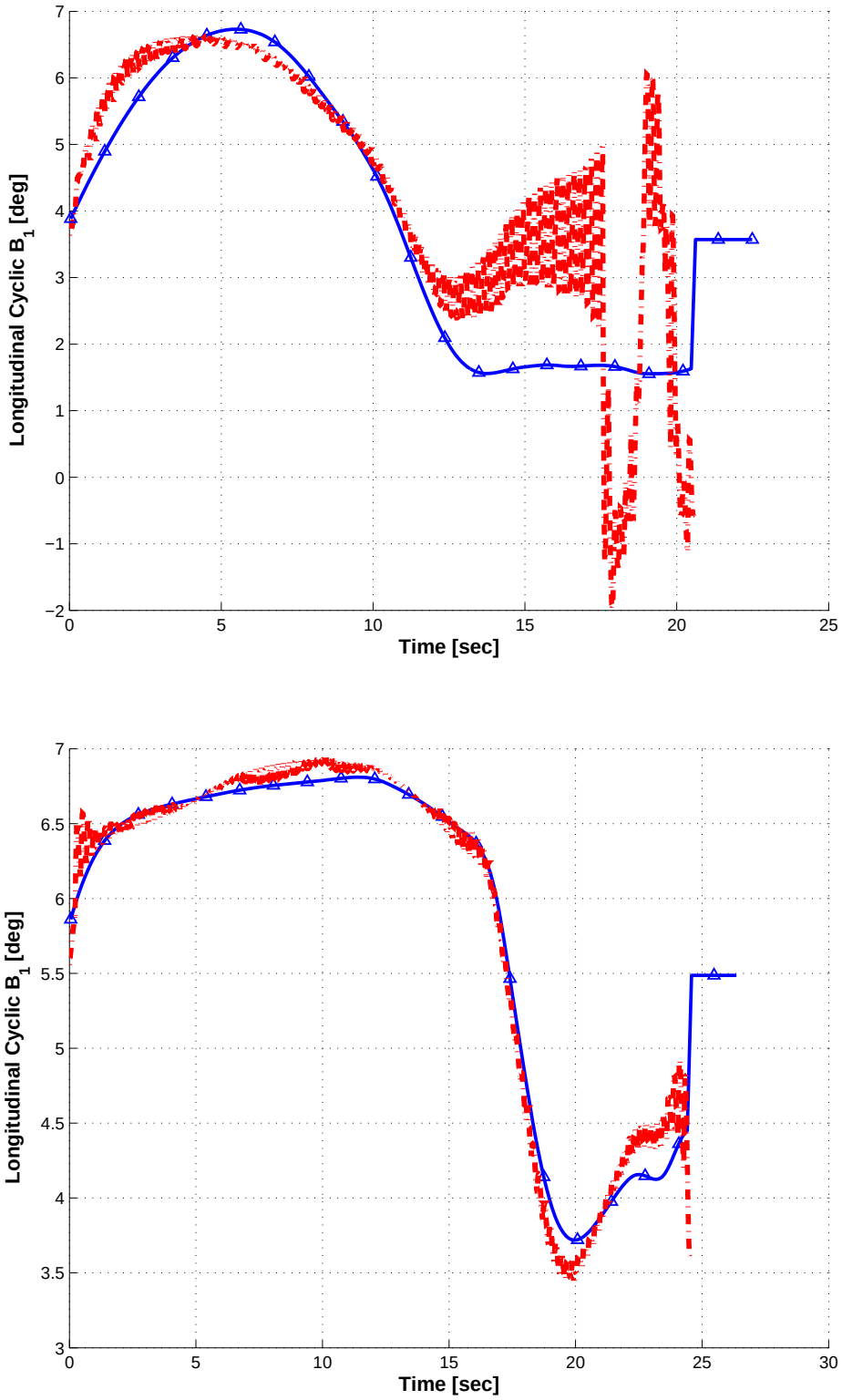

Figure 27: Helicopter obstacle avoidance maneuver with initial speed 50m/s. Longitudinal cyclic for the reduced (Δ line) and multibody (solid line) models, before (top), after (bottom) four planning / tracking steering / adaption iterations.

CHAPTER VI

OPTIMAL NEURAL NETWORK DESIGN FOR GIVEN CLASS OF PROBLEMS

In this chapter, we will try to find the optimal neural network architecture for the introduced class of problems with the information given in the previous chapters. Before designing the architecture of the neural network, the role of the entire network must be defined. The block diagram given in figure (28) is the structure of the NMPC with the feedback loop used for the applications in this study.

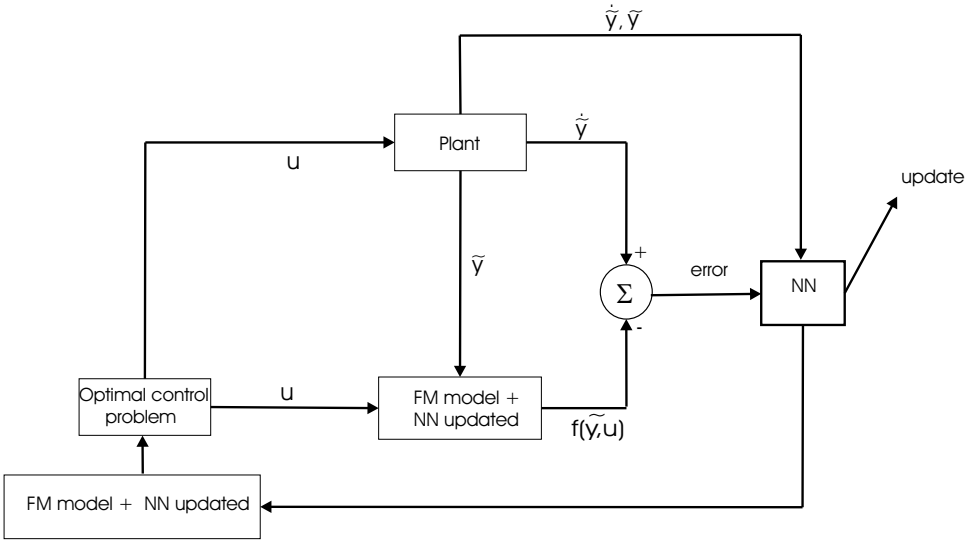

Figure 28: NMPC with the feedback loop

6.1 Designing the Architecture

This critical portion of the study can affect the performance of the application dramatically. The architecture of the neural network has a huge effect on the augmentation of the reference model. In order to get good performance in terms of small tracking errors, we have to optimize our NN.

First of all, a decision should be made on the number of hidden layers necessary for this

design. There is a consensus based on theoretical considerations that one hidden layer is sufficient [26]. Practically, most of the similar studies done so far have proven this fact by proposing a promising performance with one hidden layer architectures.

Due to this fact, it is decided that the NN will have fully connected feedforward architecture with one hidden layer as previously shown in figure (5). Lets define the structure of the layers one by one.

6.1.1 Input Layer

The input vector elements enter the network through this layer. The dimension of the information fed to the network at each iteration will determine the number of neurons that we need to use. The key idea is to determine the information that we should feed to the network to get better performance. The objective is to predict the defect of the reduced model($\boldsymbol{d}(\dot{\widetilde{\boldsymbol{y}}}, \widetilde{\boldsymbol{y}}, \boldsymbol{u})$) which is a function of full model states, derivative of states (outputs of plant) and controls. Input vector seen in equation (67) is selected in this manner.

$$\text{Input vector} = \begin{bmatrix} V_x & V_z & q & w & \theta & \theta_{mr} & B_1 & \dot{V}_x & \dot{V}_z & \dot{q} & \dot{w} \end{bmatrix}^T \tag{67}$$

When a parameter is constant during the maneuver, it provides no useful information. It can be kept as it is or can be removed from the vector space, since no weight change will be observed in contribution with that input.

One important step that should be taken here is the normalization of the inputs. Each input variable should be preprocessed so that its mean value, averaged over the entire training set, is close to zero or else it will be small compared to its standard deviation. When we consider the extreme case, where the input variables are consistently positive, the synaptic weights in the first hidden layer can only increase together or decrease together. Accordingly, if the weight vector of that neuron is to change direction, it can only do so by zigzagging its way through the error surface, which is typically slow and should therefore be avoided.

In order to accelerate the back-propagation learning process, the normalization of the inputs should also include two other measures given below:

- The input variables contained in the training set should be uncorrelated.

- The decorrelated input variables should be scaled so that their covariances are approximately equal, thereby ensuring that the different synaptic weights in the network learn at approximately the same speed.

The normalization is defined by the formula given in equation (68) where p represents the original parameter and p_n represents the normalized value of the parameter while p_{max} and p_{min} indicates the minimum and maximum values of that variable respectively, over the entire training set.

$$p_n = 2 * \frac{p - p_{min}}{p_{max} - p_{min}} - 1 \tag{68}$$

In this study, maximum and minimum values of the parameters are estimated by adding small amounts of deflection to the maximum and minimum values of the optimal solution found by using the reduced model.

6.1.2 Hidden Layer

Decision on the hidden layer of the NN is the most difficult part among the overall architecture, and for this reason there is no way to determine the best number of hidden layer neurons yet. Many heuristic techniques were suggested for finding the optimal number of neurons in the hidden layer, and several of them are currently being used. Most of them employ trial and error methods in which the NN starts with a small number of hidden layer neurons and additional neurons are added until some performance goal is satisfied. In order to do that several networks should be trained and the generalization error of each should be computed.

It is known that too many neurons degrade the effectiveness of the model. In this case, we can get low training error but still have high generalization error due to the overfitting caused by the huge number of connection weights and high variance. This is not efficient for the generalization capacity of the neural network [26].

It is also known that, too few hidden neurons may not capture the full complexity of the data because of underfitting and high statistical bias. In this case, we will get high training

error and high generalization error [26].

In real-life modelling, a small number of training examples can be available. Choosing a large number of hidden neurons in a neural network leads to undesirable consequences such as large number of connection weights in the model, long training times, local minima and small generalization capacity.

The complexities determining the optimal number of hidden neurons, includes the number of input and output units, the number of training cases, the complexity of the function or classification to be learned, the architecture, the type of hidden activation function and the training algorithm.

As it is mentioned above, there are some rules that have been proposed to find the optimal number of hidden layers. They include:

- Hidden layer size should be somewhere between the input layer size and output layer size.
- This formulation is proposed for the calculation of the number of hidden layer neurons: (Number of hidden layer neurons + number of output layer neurons)*(2/3).
- It should never be more than twice as large as the input layer.

However, we can not say that these rules are general and can be applied to any problem. These can be helpful in finding the starting point. Reference [29] proposes a statistical procedure for determining the optimal number of hidden neurons.

The other important issue is the selection of the activation function. The sigmoid activation function defined in figure (2) ranges from 0 to 1. It is sometimes desirable to have the activation function range from -1 to 1 in which case the activation function assumes an antisymmetric form with respect to the origin. For the corresponding form of the sigmoid function, hyperbolic tangent function issued by equation (69).

$$\varphi(\nu) = \tanh(\nu) \tag{69}$$

For the convenience of the problem hyperbolic, tangent function is chosen as an activation function where we can also have the negative range. In Matlab this function is activated by 'tansig' command.

The optimal number of hidden layer neurons are discussed in section 3.

6.1.3 Output Layer

The output layer of the network is concerned about the defect between two models. This defect is defined in the accelerations domain by the formula (25d). Due to this reason, it will be useful to define the the output vector in the following way as seen in equation (70) which correspond to the predicted defect.

$$\text{Output vector} = \boldsymbol{d}(\tilde{\tilde{\boldsymbol{y}}}, \widetilde{\boldsymbol{y}}, \boldsymbol{u}) \tag{70}$$

The activation function used in this layer is a linear function which is activated by 'purelin' command in Matlab.

Another important step that should be taken before starting training is the initialization of the synaptic weights. Since there is no prior information available at the beginning of the training, the weights and biases of the NN are initialized. In the beginning, initialization is done by randomly selected values, small enough to guarantee not to affect the system's behavior. Also an evolutionary initialization methodology is proposed in reference [28].

6.2 Strategies for Training

The neural network architecture is defined in the previous section as 11-h-3 fully connected feedforward network where h represents the optimal number of hidden layer neurons that will be determined.

The next step is defining the strategy, used, in terms of algorithm, training mode and learning rate.

6.2.1 Algorithm and Training Mode

As explained in previous chapters, we are using receding horizon methodology to provide feedback to the system by the repetitive solution of the problem. Also during this phase, in

order to improve the fidelity of the prediction, reduced model is adapted iteratively. After steering the MB model, sufficient information is gathered to adapt the FM model until the maneuver ends. The procedure includes more than the incremental mode, since there is more than one example set for each steering window. In this part, batch training is preferred specifically for the steering windows during the maneuver. Both batch and incremental modes are taking place in the training procedure. The overall training procedure is based on on-line training methodology, but the combination of training modes is referred to *batch-incremental mode* for this class of applications.

In large applications, batch learning is experienced to be rather infeasible and instead on-line learning is employed. It fits well into more natural or life-long learning since during the on-line training, the learner receives new information at every moment and should adapt to it, without having large memory for storing old data. With batch learning by construction, changes typically go undetected and rather bad results can be obtained, since we are likely to average over several rules , whereas on-line training, if operated properly will track the changes and yield good approximation [25].

The training algorithm used in this study is the simplest gradient descent methodology which is steepest descent. For the on-line training applications steepest descent can be useful since the formulation of adaption is simple and this makes it convenient to track the behavior of every variable during the process.

6.2.2 Learning Rate Strategy

The efficiency of the on-line learning is highly dependent on the learning rate. Approximated value for the learning rate η, depending on time should be found. Practically, if constant η is used, too small values will cause slow learning and is therefore not useful, too large η spoils the convergence of the learning [25]. Trade-off between learning speed and accuracy can be done in this case [27].

Constant Learning Rate

A feasible fixed value for η cannot be established priori because of primary problem in

simulated surface travel, namely overshoot of the goal. Fixed value for η implies a finite step length will be taken at every point on the error weight surface that is not extremum. The length of step for a given nonzero gradient can always be shown to overshoot the unknown goal state if the gradient occurs near enough to the goal as shown in figure (29).

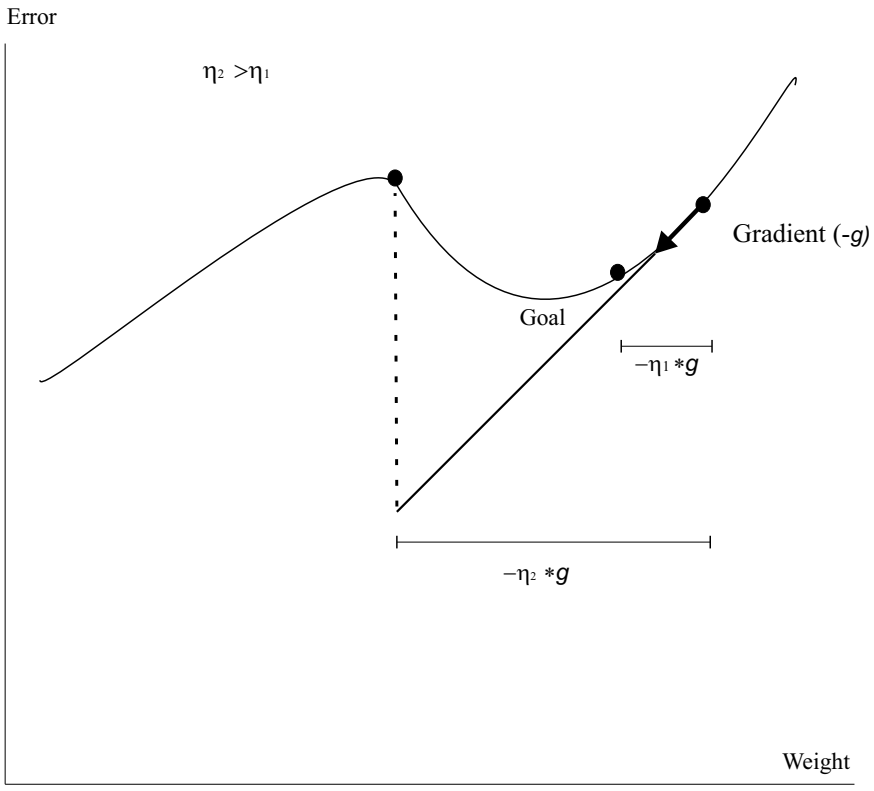

Figure 29: A step overshooting the goal.

In standard back-propagation algorithms, η is empirically determined. An initial arbitrary value is used and if this fails, further values are tried [23].

For this application, a constant value for learning rate is set and program launched. If the unstable behavior of the system is observed, then this value is decreased. In this sense, an optimal constant value found as 0.02 for the learning rate.

Adaptive Learning Rate

The idea of adaptively changing η is called adaptive learning rate or learning of the learning rule. By making the η adaptive we can reduce the average training time with the faster convergence. The approach here will be to use an optimal safe step size. Detailed

information is given about finding the optimal step size is given in reference [23].

Adaptive learning strategy has been tried in some different ways in order to get better performance and faster convergence.

Normally, Matlab has a standard training function for adaptive learning rate which is based on the rule that; if the mean squared error (mse) between the desired output and network output is decreased after the training, then the learning rate is increased by 5%, otherwise it is decreased by 30%.

In our case, primarily we tried to decrease the error in all dimensions, rather than the rms error. This is done by checking every component of the output vector. Decrement on the error of every component is the necessity to approve the improvement of the NN. This means that we can increase the learning rate little bit more and shoot to the goal with a slightly larger step. Only one error component increase is enough to be considered as non-improvement and η is decreased in order to shoot with a smaller step. Furthermore, if an improvement is measured, then synaptic weights and biases are updated otherwise old weights are kept and moved to the next step.

Also an upper bound is set for learning rate in order not to shoot too far away from the goal. However, this adaptive methodology caused some problems such that after a while during the iteration, learning rate hit level zero and stayed there until the end of the iteration. Due to this reason, the strategy is switched to the original mse error check.

Everything mentioned above remained same, with the only change being the standard mse error check.

For this case, several learning rate upper bounds have been tried, but in every case initial learning rate increased to its upper limit and stayed there during the rest of the iterations. This came to the same point as using a constant learning rate. It is also observed that the upper bound value for the learning rate cannot be greater than 0.08 since the system tends to show unstable behavior above that value.

The optimal value for upper bound is the 0.02 where the best performance was achieved. As a conclusion, the adaptive learning rate did not pay off for these kind of problems. It always tended to hit the upper bound and stayed there acting like a constant learning rate.

The constant learning rate is preferred over the adaptive strategy for these type of studies.

6.3 Decision on Optimal Architecture

So far the input and the output layer of the architecture has been built up. Now, the number of the hidden layer neurons must be determined in order to get the optimal performance from the network. To do this, we used the most common methodology, trial and error. Regarding the complexity of the problem, the initial number of neurons selected was 20 which is high enough depending on the advice given in the previous chapter. Considerably good results have been achieved with 20 hidden layer neurons, so all learning rate strategy was developed with this architecture. The number of hidden layer neurons were decreased sequently from 20 to 15 and finally 10. Almost no change was observed in the performance of the algorithm. However, below 10 hidden layer neurons learning of the network started to slow down. As an example, for the 6 hidden neurons case, it is observed that two more replannings are needed to arrive the same performance of 10 hidden neurons.

Consequently, these results show that a robust neural network is created. Based on this study, we can conclude that the optimal architecture can be 11-10-3 fully connected feedforward network for the given class of problems.

As explained in detail in the previous section, constant learning rate is preferred over adaptive. The optimal value experienced is to be 0.02.

6.4 Contributions of neural network to reduced model equations

Now, lets point out the contributions of the NN to the dynamic equations of the reduced model. Since the reduced model parameters are the weights and biases of the NN, by looking at the behavior of those parameters during the entire process, we can deem on which parameters are most effective. The figures given below provides the important weight and bias changes observed during the process.

As seen from the figures, contribution of the weights of pitch rate are highly effective. Also in the output layer, angular acceleration component of the defect is large. It is safe

to say that the main adjustment on the reduced model equations is on the pitch moment equation given in reference [3].

The largest values of the weights were observed in the output layer biases. Also hidden layer biases change more rapid compared with the others. Some other figures about the weight analysis provided in the appendix B.

Figures (30-33) show that most of the weights of the neural network almost converged to a specific value. This matches with the fact that no more improvement is possible after sufficient replanning / multiple tracking and steering iterations are done.

6.5 Effect of neural network and NMPC methodology to the system stability and convergence

In the case of full match between reduced and full models, non-linear model predictive control methodology can ensure the stability of the closed-loop system and convergence to the reference trajectory, under the assumptions of infinite prediction horizon in the tracking phase.

However, in practice, the methodology is based on the use of finite prediction and control horizons. It is also possible to achieve closed-loop stability with finite horizon, for example by introducing some stability constraints. In this case, the system can be required to reach the target solution in finite time by enforcing proper constraints in solving the tracking problem.

Moreover, in this study, we have a reduced model which is a rough representation of the plant. In this sense, the outputs of the plant and reference model will be different and these have to be matched by reducing the mismatch between them. For example, the reference model can be augmented by using a neural network. The universal approximation property of neural networks [12] ensures that the reconstruction error in the equation (37) can be bounded in a small region, so that the defect of the reference model can be captured with a desired accuracy. Therefore, given a particular structure of the network, a set of parameters of the adaptive element exist that give the desired accuracy of the reduced model. It is also mentioned before that, above a critical value of the learning rate, the algorithm can show

unstable behaviors. In this research, we didn't investigate the effect of the tuning algorithm to the stability of the non-linear predictive controller.

In this study, the adaptive element of the reduced model is represented by the neural network and it is used in solving the optimal control problems in both planning and tracking levels, so the system behavior is highly based on the correct augmentation of the reference model. Although the adaptive algorithm in this research cannot be proved analytically to be efficient in finding the combination of parameters, there are some published theoretical results show that an adaptive rule formulated in a correct way which guarantees the convergence of the neural network parameters, can be constructed in some cases [30, 31].

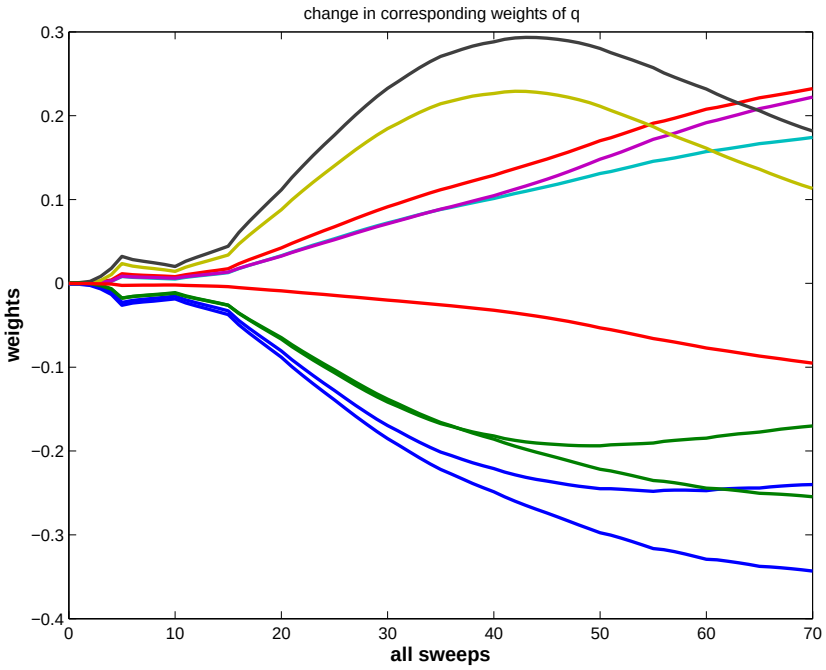

Figure 30: Weight contribution q

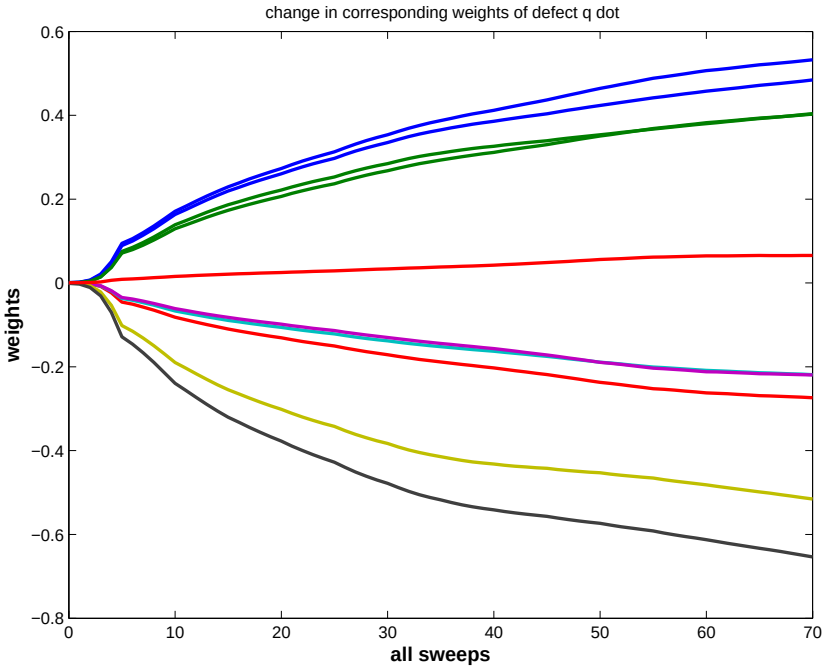

Figure 31: Weight contribution from angular acceleration component of defect

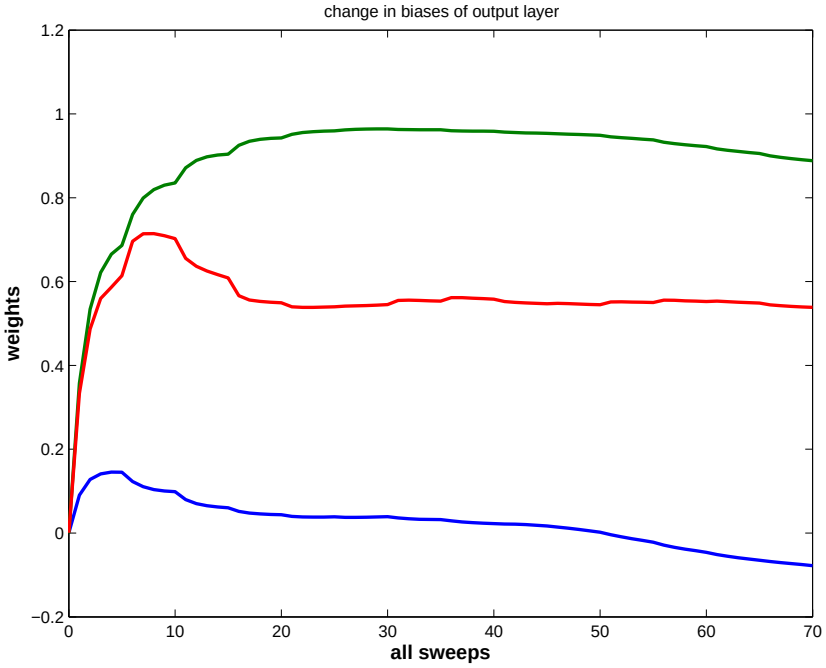

Figure 32: Weight contribution output bias

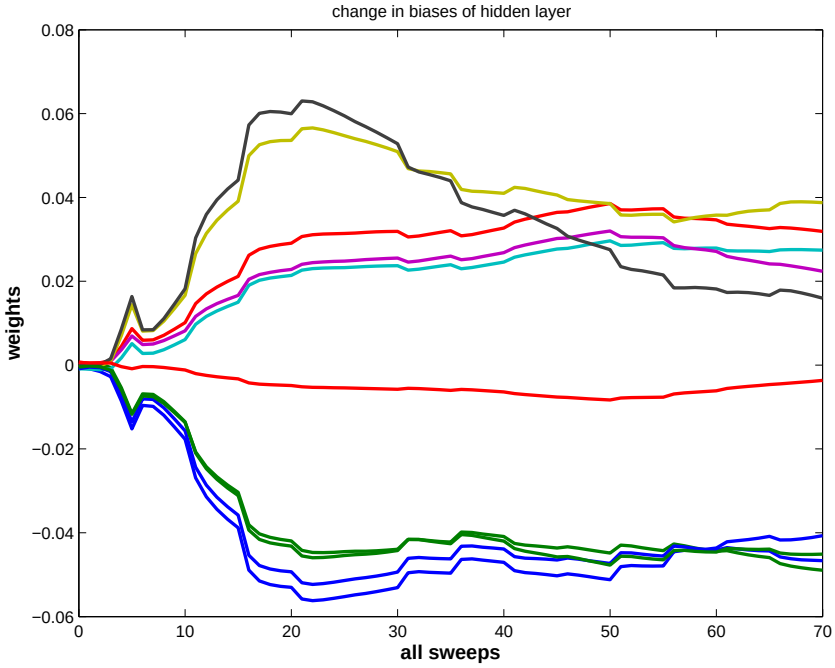

Figure 33: Weight contribution hidden bias

CHAPTER VII

CONCLUSION

In this study, a detailed procedure for the simulation of maneuvers with multibody models has been proposed. Using a neural network as an adaptive controller results in a good tracking performance and also provides the use of the reduced model for the path planning phase. Some advantages of this procedure are as follows:

- We can maneuver vehicle models that have arbitrary complexity with large number of degrees of freedom.
- We can compute compatible optimal trajectories with vehicle dynamics based on a given task.
- Using non-linear model predictive controller in solving the tracking problem provides improved tracking performance with respect to the other control strategies.
- The same software can be used for planning and tracking phase since they are both based on the optimal control problem of a reduced model.
- Using neural network as an adaptive element increases the predictive capabilities of the reduced model in a robust way.

It is experienced that the adaptive learning rate strategy doesn't pay off for the given class of problems, so constant learning rate could be preferred during the training procedure. This constant rate should be chosen small enough in order not to overshoot the goal in the error surface which can cause adaption of the model badly and can result in instabilities of the reference solution.

The proposed adaptive element (neural network) ensured its robustness by providing good results for different architectures. Basics of the given architecture can be applied to further studies.

The results of the obstacle avoidance problem proved that we can decrease the mismatch between two models and come up with a close match between steering and tracking trajectories by the proposed adaptive NMPC methodology in this study.

Pre-training (off-line training) is still an option which can be useful to start with better initialized values of neural network parameters. However, ranges of states and controls must be well determined in pre-training, in order to capture the bounds of the problem. This is still an important issue because of high computational costs in creating pre-training data.

As a future work, other training algorithms mentioned in chapter(2) can be tried to increase the performance of the neural network.

Furthermore, adaptive learning rate strategy tried in this study can be augmented with different possible ideas that prevent the learning rate hitting zero level.

APPENDIX A

RESULTS FOR THE EXAMPLE WITH INITIAL SPEED 30M/S

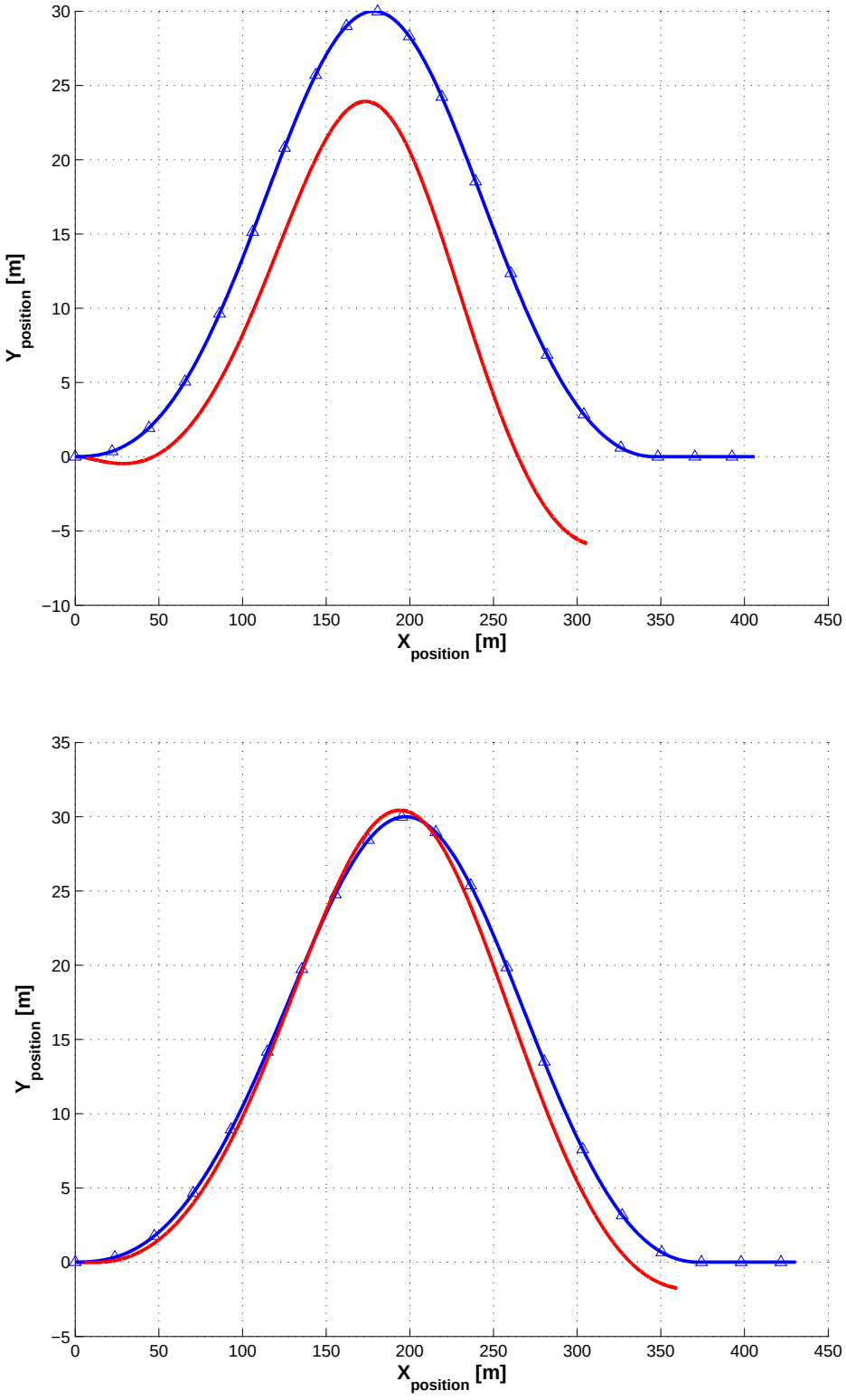

Figure 34: Helicopter obstacle avoidance maneuver with initial speed 30m/s. Trajectory flown by the reduced (Δ line) and multibody (solid line) models, before (top), after (bottom) fourteen planning / tracking steering / adaption iterations.

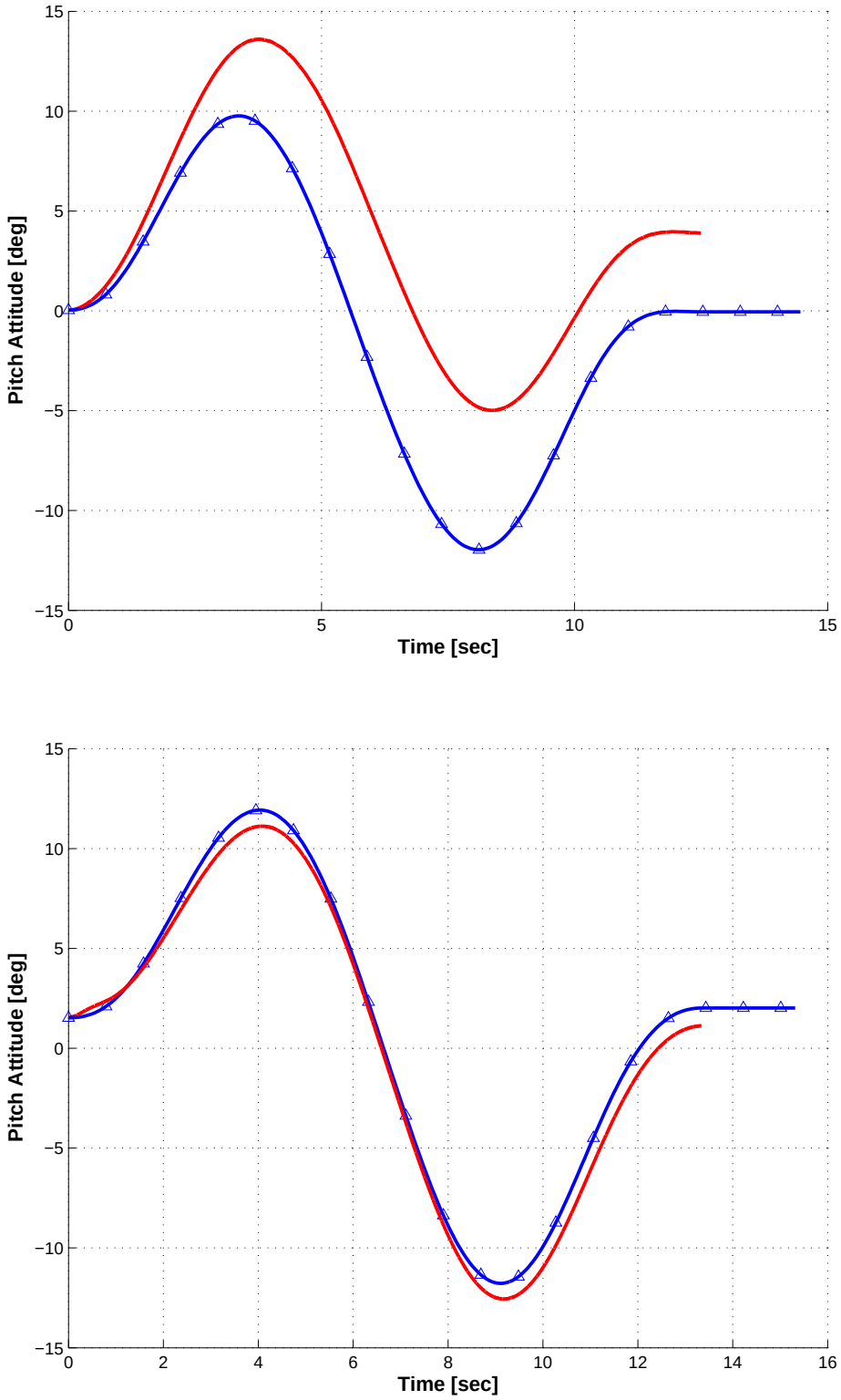

Figure 35: Helicopter obstacle avoidance maneuver with initial speed 30m/s. Fuselage pitch for the reduced (Δ line) and multibody (solid line) models, before (top), after (bottom) fourteen planning / tracking steering / adaption iterations.

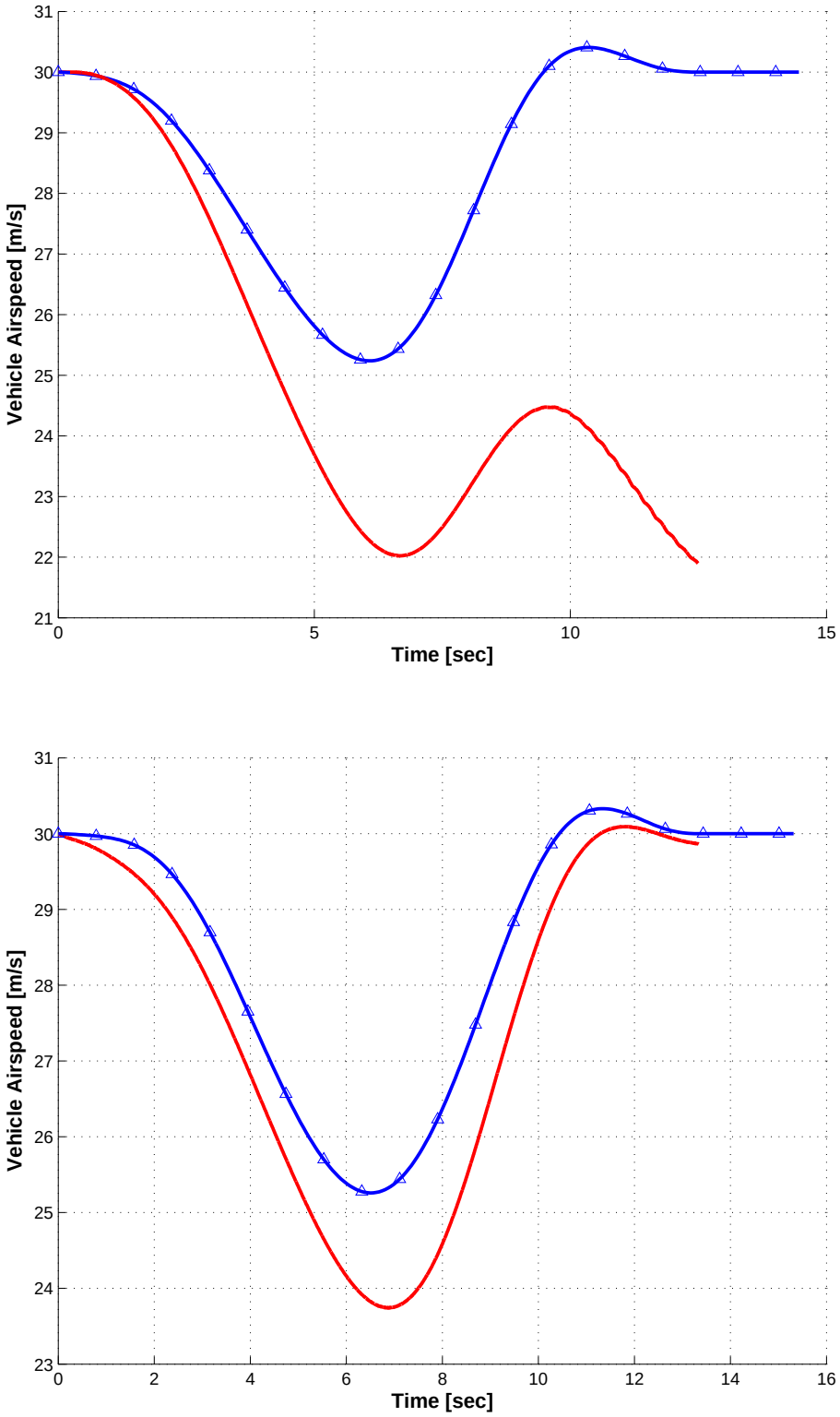

Figure 36: Helicopter obstacle avoidance maneuver with initial speed 30m/s. Rotorcraft speed for the reduced (Δ line) and multibody (solid line) models, before (top), after (bottom) four planning / tracking steering / adaption iterations.

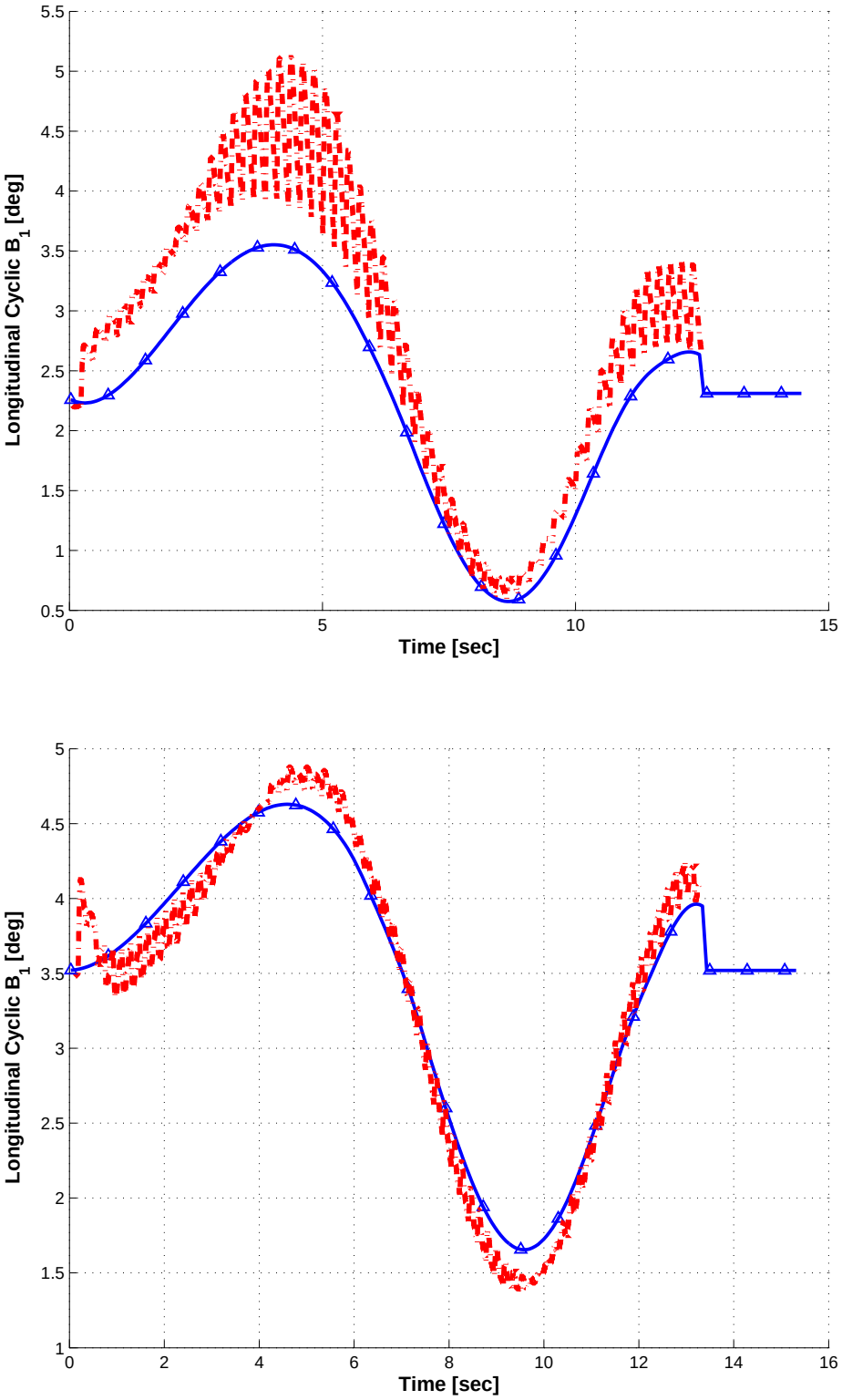

Figure 37: Helicopter obstacle avoidance maneuver with initial speed 30m/s. Longitudinal cyclic for the reduced (Δ line) and multibody (solid line) models, before (top), after (bottom) four planning / tracking steering / adaption iterations.

APPENDIX B

WEIGHT CHANGES FOR THE 50 M/S CASE

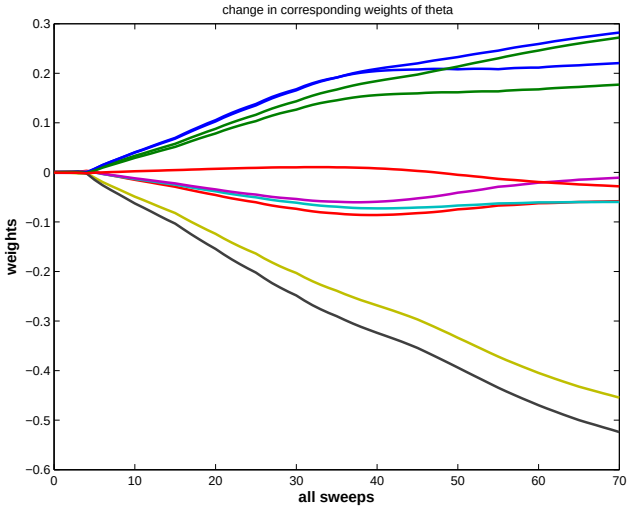

Figure 38: Weight contribution theta

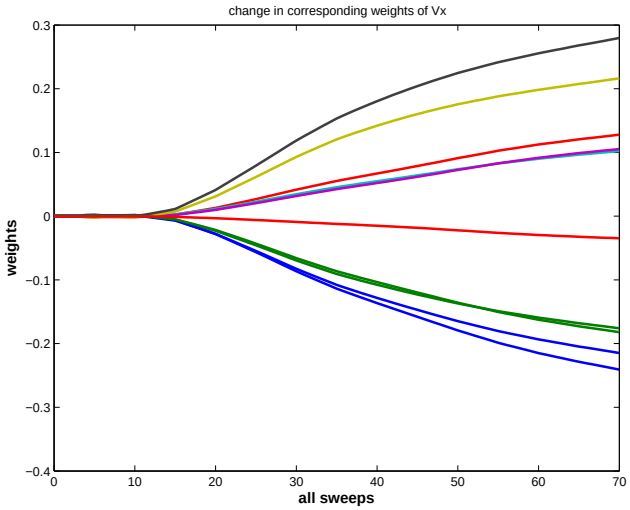

Figure 39: Weight contribution Vx

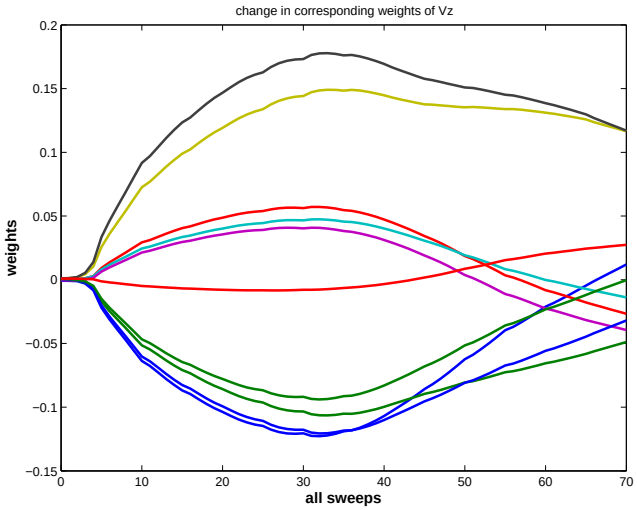

Figure 40: Weight contribution Vz

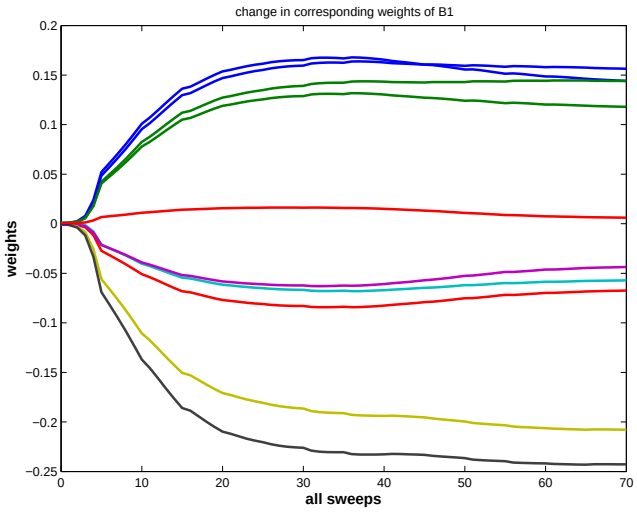

Figure 41: Weight contribution B1

REFERENCES

[1] C.L. Bottasso, Chong-Seok Chang, A. Croce, D. Leonello, L. Riviello, Adaptive planning and tracking of trajectories for the simulation of maneuvers with multibody models, Invited paper, to appear in special issue "Computational Multibody Dynamics' of Computer Methods in Applied Mechanics and Engineering.

[2] C.L. Bottasso, A. Croce, D. Leonello, L. Riviello, Unsteady trim for the simulation of maneuvering rotorcraft with comprehensive models, Journal of the American Helicopter Society (2004) under review.

[3] C.L. Bottasso, A. Croce, D. Leonello, L. Riviello, Optimization of critical trajectories for rotorcraft vehicles, Journal of the American Helicopter Society (2004) accepted, to appear.

[4] A. Croce, Trajectory optimization of multibody systems with applications to the maneuvering flight of rotorcraft vehicles, Ph. D. Thesis, 2004

[5] A. Peters, D. Barwey, A general theory of rotorcraft trim, Mathamatical Problems in Engineering, 2, 1996, pp. 1-34

[6] O.A. Bauchau, C.L. Bottasso, Y.G. Nikishov, Modeling rotorcraft dynamics with finite element multibody procedures, Mathematics and Computer Modeling 33 (2001), pp. 1113-1137

[7] M.J. Bhagwat, J.G. Leishman, Stability consistency and convergence of time marching free-vortex rotor wake algorithms, Journal of the American Helicopter Society 46 (2001), pp. 59-71

[8] O.A. Bauchau, C.L. Bottasso, L. Trainelli, Robust integration schemes for flexible multibody systems, Computer Methods in Applied Mechanics and Engineering 192, (2003) pp. 395-420.

[9] C.L. Bottasso, O.A. Bauchau, On the design of energy preserving and decaying schemes for flexible, non-linear multibody systems, Computer Methods in Applied Mechanics and Engineering, 169 (1999), pp. 61-79.

[10] W. Johnson, Helicopter Theory. Dover Publications, New York, 1994.

[11] R. Findeisan, L. Imland, F. Allgower, B.A. Foss, State and output feedback non-linear model predictive control: an overview, European Journal of Control 9 (2003), pp. 190-206.

[12] K. Hornik, M. Stinchombe, H. White, Multi-layer feed-forward networks are universal approximators, Neural Networks 2 (1989), pp. 359-366.

[13] L. Fausett, Fundementals of Neural Networks, Prentice Hall, New York, 1975.

[14] S. Haykin, Neural Networks a Comprehensive Foundation, Prentice Hall, New Jersey, 1999.

[15] J.T. Betts, Practical Methods for Optimal Control Problem Using Non-Linear Programming, SIAM, Philadelphia, 2001.

[16] D.A. Peters, C.J. He, Finite state induced flow models. Part 2: three dimensional rotor disk, Journal of Aircraft 32 (1995), pp. 323-333.

[17] C.L. Bottasso, A new look at finite elements in time: a variational interpretation of Runge-Kutta methods, Applied Numerical Mathematics 25 (1997), pp. 355-368.

[18] K.S. Narendra, K. Parthasarathy, Using neural networks, IEEE (1990).

[19] K.S. Narendra, A.M. Annaswamy, Stable Adaptive Systems, Englewood Cliffs, Prentice-Hall, New Jersey (1989).

[20] M.M. Gupta, D.H. Rao, Neuro Control Systems:Theory and Applications, IEEE Neural Network Council, IEEE Press, New York, (1994).

[21] B. Kouvaritakis, M. Cannon, Nonlinear Predictive Control: Theory and Practice, IEEE Control Engineer Series 61, The Institution of Electrical Engineers, London, (2001).

[22] A.M.S Zalzala, A.S. Morris, Neural Networks for Robotic Control: Theory and Applications, Ellis Horwood, London, 1996.

[23] M.K. Weir, A method for self-determination of adaptive learning rates in back-propagation, Neural Networks 4, (1991), pp. 371-379.

[24] A. Abraham, B. Nath, An adaptive learning framework for optimizing artificial neural networks, ICCS 2001, LNCS 2074, (2001), pp. 171-180.

[25] N. Murata, M. Kawanbe, A. Ziehe, K.R. Muller, S. Amari, On-line learning in changing enviroments with applications in supervised and unsupervised learning, Neural Networks 15, (2002), pp. 743-760.

[26] Z. Boger, R. Weber, Finding an optimal artifical neural network topology in real-life modeling: Two approaches, ICSC Symposium on Neural Computation (2000), Article no: 1403/109.

[27] S. Amari, Theory of adaptive pattern classifiers, IEEE Transactions in EC, (1967) 16(3), 299-307.

[28] C. Gegout, Improvement of gradient descent based algorithms training multilayer perceptrons with an evolutionary initialization, Neural Networks and Computational Learning Theory (1995), Technical Report NC-TR-95-028 .

[29] I. Rivals, L.Personnaz, A statistical procedure for determining the optimal number of hidden neurons of a neural model, Second International Symposium on Neural Computation (2000), Berlin.

[30] A.J. Calise, R.T. Rysdyk, Nonlinear adaptive flight control using neural networks, IEEE Controls Systems Magazine, December 1998 18(6), 14-25,

[31] E.N. Johnson, S.K. Kannan, Adaptive flight control for an autonomous unmanned helicopter, AIAA Guidance, Navigation and Control Conference, number AIAA-2002-4439, Monterey, CA, August 2002.