

# Real-Time Geometrical Approximation of Flexible Structures using Neural Networks

Karl Mathia and Kevin Priddy, Ph.D.

Accurate Automation Corporation  
7001 Shallowford Road  
Chattanooga, TN 37421

**Abstract.** This study demonstrates the potential of artificial neural networks for the geometrical approximation of flexible structures. On-line modeling of the deformation and dynamics of flexible structures can improve the control and performance of systems such as airplane wings, rotor blades of helicopters, large articulated space structures, and robots with flexible links or joints. Here a neural model that approximates the deflection of such structures is developed. Real-time modeling is provided by a specialized neural network processor. We demonstrate this concept using the model for the nonlinear deflection of an viscoelastic airplane wing.

## 1 Problem Context

The need for light-weight structures and the advent of new composite materials, combined with increased performance requirements, challenge control engineers in a broad spectrum of industrial applications. Many problems arise from undesired dynamics of systems, introduced by the flexing of substructures. All stages of control system design are affected by these complex dynamics, i.e. plant modeling, model analysis and controller design. Examples are the tracking control of robots with flexible joints and links [12], point and shape control of articulated space structures [11], and the vibration suppression of high-rising buildings [1]. For certain aeronautical applications, the problem reduces to the modeling and control of a one-dimensional, flexible beam, like the main rotor of helicopters [5]. In these cases the problem is to *control* the dynamics of the flexible structure itself, while the present study is concerned with the real-time *modeling* of such structures. The approach described here applies to systems which comprise one or more flexible structures, and whose dynamics are affected by these substructures, and whose performance specifications require the on-line modeling of the flexing dynamics to be included into the control system.

The deflection of airplane wings usually changes as a function of the flight situation, as symbolized in Figure 1. The relaxed wing, with no forces acting on it, defines the  $x$ -axis of the virtual reference system and the deflection is zero in this configuration,  $y = 0$ . During high-g maneuvers in

flight, the resulting forces deform the wing, such that  $y \neq 0$ . The adaptation of flight controllers to this changing aerodynamic characteristic was recently considered [4]. Gain scheduling or an adaptive control law are possible control concepts, both of which would require the real-time monitoring of the nonlinear deflection. The neural model developed in the present work is a means to monitor the wing deflection. It comprises two different ANNs: a functional link network is used as the deflection model, whose free parameters are continuously updated by a multi-layer preceptron, based on wing curvature measurements. Real-time requirements are met by the implementation of both networks on a specialized neural network processor.

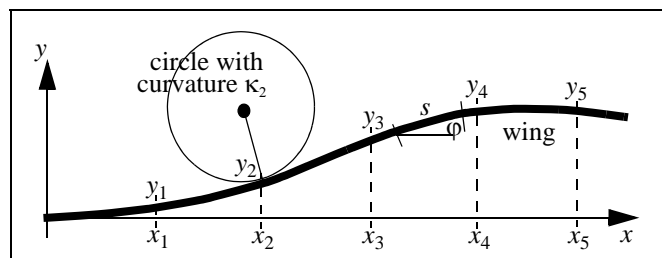


Figure 1. Symbolized vertical deflection of an airplane wing.

## 2 Multilayer Perceptron and Functional Link Network

The neural wing deflection model employs two ANN paradigms, a functional link network and a multilayer perceptron. Both networks have a feedforward architecture, learn from examples and are often used as function approximation networks.

### 2.1 Architecture

The *multilayer perceptron* (MLP) is a widely used ANN, in particular since the backpropagation learning algorithm was popularized in 1986 [7]. It can have one or more hidden layers of artificial neurons, or processing elements (PEs), and is usually used as a general purpose function synthesis network, since it can accurately approximate a broad class of

nonlinear functions. For the present work, a single-hidden layer MLP was used (Figure 2). The hidden PEs employ sigmoid transfer functions  $g(\cdot)$ , while the output PEs have linear transfer functions. The bias element provides a constant unity input. The map of this MLP architecture is given by

$$\mathbf{y} = \mathbf{f}_{\text{MLP}}(\mathbf{x}) = \mathbf{W}_2 \cdot \mathbf{g}(\mathbf{W}_1 \mathbf{x} + \mathbf{w}_b), \quad (1)$$

where  $\mathbf{W}_1$ ,  $\mathbf{W}_2$  are the weight matrices of hidden layer and output layer, respectively,  $\mathbf{w}_b$  is the weight vector from hidden layer to bias element, and the function vector  $\mathbf{g}(\cdot)$  represents the sigmoids in the hidden layer. The connection weights are the free parameters of the MLP and are specified during learning.

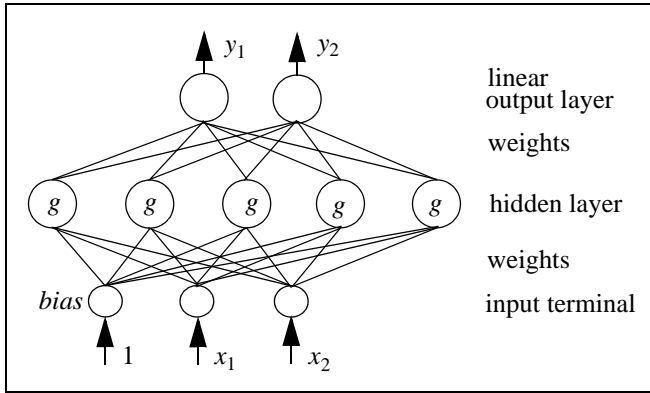


Figure 2. A multilayer perceptron.

The *functional link network* (FLN) can employ a variety of feedforward architectures [6]. In addition to the original inputs  $x_i$ , the FLN provides functional links of these  $x_i$  to the subsequent layer, therefore enhancing the input space with additional dimensions. Figure 3 shows a 2-input 2-output FLN with functional links  $f_1$ ,  $f_2$ , and  $f_3$ . The increased input dimensionality provides the hyperplanes of the network with a greater ability to approximate a function or to separate the input space. This can make hidden layers unnecessary and improves the learning performance of the FLN. Thus the ease of implementation is another potential advantage of FLNs.

A special case of the FLN is the joint activation network (JAN) [3], another variant is the polynomial network [10]. Both networks implement parameterized multivariable polynomials. If the functional links  $f_1$ ,  $f_2$ , and  $f_3$  are multipliers, the network in Figure 3 has the form

$$\begin{aligned} \mathbf{y} &= \mathbf{f}_{\text{FLN}}(\mathbf{w}, \mathbf{x}) \\ &= w_0 + w_1 x_1 + w_2 x_2 + f_1(x_1) + f_2(x_2) + f_3(x_1, x_2) \\ &= w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2. \end{aligned} \quad (2)$$

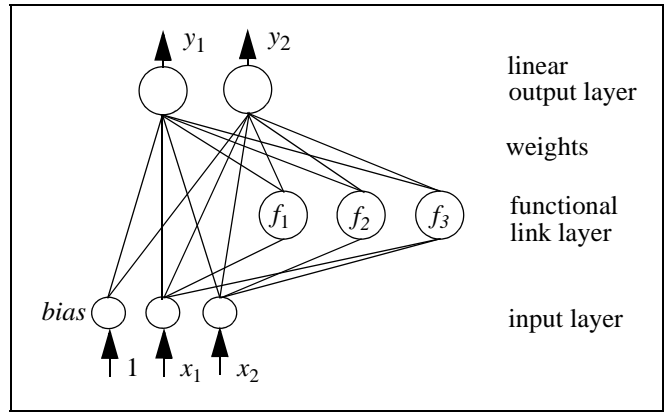


Figure 3. Functional link network.

A JAN was used for the present work, and the notation of functional link networks during the following discussion refers to the JANs.

## 2.2 Learning

The learning algorithm commonly used to adapt the MLP connection weights is the generalized delta rule. The algorithm minimizes the total squared error  $E$ , the difference  $\delta$  between actual output  $\mathbf{y}$  and desired output  $\mathbf{y}_d$ ,

$$E = \frac{1}{2} \cdot \sum (y - y_d)^2 = \frac{1}{2} \cdot \sum \delta^2, \quad (3)$$

taken over all output PEs. The MLP learns from examples by updating its connection weights according to their contribution to  $E$ , which is propagated backwards through the network. The generalized delta rule therefore is also called the error backpropagation algorithm [7]. It is based on the gradient descent method, and, at step  $k$  of the iterative learning process, the weight vector  $\mathbf{w}$  of each PE is updated according to

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \Delta \mathbf{w}_k = \mathbf{w}_k - \eta \cdot \nabla_{\mathbf{w}} E_k. \quad (4)$$

Since  $E$  is to be minimized, the negative gradient,  $-\nabla_{\mathbf{w}} E_k$ , determines  $\Delta \mathbf{w}$ , the adjustment of  $\mathbf{w}$ . The learning rate  $\eta$  is a small constant scalar, usually less than 1.

The backpropagation algorithm can also be applied to train FLNs. If a feedforward network has no hidden layers, like the FLN used here (Figure 3), the computation of the gradient in Equation 4 is facilitated and the generalized delta rule reduces to the simple delta rule [6]. The rapid training adjusts the weight vector  $\mathbf{w}$  of all FLN PEs such that the total squared error in Equation 3 is minimized. We found the joint activation network [3], which is somewhat related to the polynomial network [10], well suited for the present deflection modeling problem.

### 3 Designing the Neural Model

During flight it is technically difficult to measure the vertical deflection of the viscoelastic wing relatively to some reference system. In the present simulation study the wing curvatures were measured and processed by an MLP and an FLN to approximate the wing deflection.

#### 3.1 Concept

The concept of the neural deflection model is shown in Figure 4. The wing curvatures are measured at reference locations along the wing's leading edge. The small deflection angles allow us to assume these sensor locations  $x_i$  are on the  $x$ -axis (Figure 1). The curvature measurements,  $\kappa_j$ , are sufficient input to the neural wing deflection model to approximate the wing deflection  $y$ . An MLP maps the  $\kappa_j$  to the corresponding parameters of the polynomial deflection model, the weight vectors  $w$  of the FLN neurons. The FLN then approximates the deflection vector  $y$ , relative to the  $x$ -axis (the undeformed wing configuration). It is interesting to notice that the model parameters  $w$  and the variable input  $x$  switch their roles in this concept:  $x$  is constant, while  $w$  is the variable input vector.

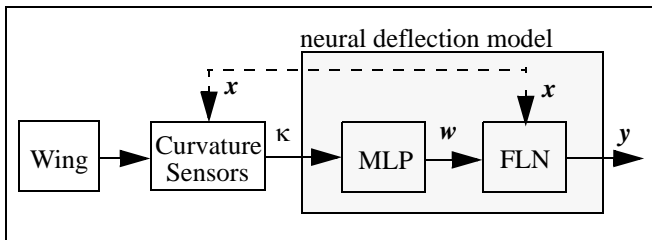


Figure 4. The neural deflection model.

With  $\kappa \in \mathfrak{R}^m$ ,  $w \in \mathfrak{R}^n$ ,  $x \in \mathfrak{R}^m$ ,  $y \in \mathfrak{R}^m$ , the model design includes the following steps:

1. Choice of an FLN architecture for the mapping  $f_{\text{FLN}}: \mathfrak{R}^m \rightarrow \mathfrak{R}^n$ ,  $y = f_{\text{FLN}}(w)$ .
2. Choice of an MLP architecture for the mapping  $f_{\text{MLP}}: \mathfrak{R}^m \rightarrow \mathfrak{R}^n$ ,  $w = f_{\text{MLP}}(\kappa)$ .
3. Acquisition of the training set: collect data pairs  $\{\kappa_j, y_j\}$ ,  $j = 1, \dots, N$ , for  $N$  wing deflections.
4. Training the FLN: adapt weight vectors  $w_j$  such that all deflections  $x_j \rightarrow y_j$  are approximated. Save all  $w_j$ .
5. Training the MLP: learn the map from sensory inputs to the weights learned by the FLN:  $\kappa_j \rightarrow w_j$ .

The network architectures in steps 1 and 2 are discussed in Section 2. The MLP used here has  $m = 5$  inputs, 20 hidden PEs with sigmoid transfer functions, and  $n = 4$  linear output PEs (Figure 2). All layers were fully connected. The sin-

gle-input single-output FLN used here is a special case of the joint activation network in Figure 3, implementing the polynomial

$$y(x) = w_4 x^4 + w_3 x^3 + w_2 x^2 + w_1 x + w_0, \quad (5)$$

where the cross-coupling terms are omitted. After training this FLN was cloned five times. The clones were organized in a parallel architecture, so that one network was available for each sensor input. Steps 3 to 5 above, i.e. the collection of training data and neural network training, are discussed in more detail in the following.

#### 3.2 Training Data

An essential part of the design process is training the MLP and FLN to approximate the desired mappings. A 'good' training set is crucial for successful training, and must contain a sufficient number input/output pairs. For the present application; example triplets  $\{\kappa_j, w_j, y_j\}$ ,  $j = 1, \dots, N$ , are needed. The set of weight vectors  $w_j$  will be determined by the FLN. The 'sensor' locations  $x_i$  are fixed. Thus, initially it remains to collect  $\{\kappa, y\}$  pairs. For the geometrical approximation of a viscoelastic airplane wing, five curvature measurements and a 5th-order FLN (4 weights) achieved sufficient accuracy.

The curvature  $\kappa$  of a 2-dimensional curve is defined as the change of a unit tangent vector along the curve. By definition, the direction of this change is not of concern, and  $\kappa$  is given by the absolute value [9],

$$\kappa = \left| \frac{d\varphi}{ds} \right| = \left| \frac{\frac{d\varphi}{dx}}{\frac{ds}{dx}} \right| = \frac{\left| \frac{d^2 y}{dx^2} \right|}{\left[ 1 + \left( \frac{dy}{dx} \right)^2 \right]^{2/3}}, \quad (6)$$

where  $s$  is a line segment of the curve, and  $\varphi$  is the angle between  $x$ -axis and that line segment (Figure 1). The curvature measurements of the wing were simulated using a mapping of the form in Equation 6. The polynomial was used to fit a series of simulated wing deflection data. Equation 6 and Equation 5 were applied to  $N = 1000$  sets of deflection data, which gave 1000  $\{\kappa, w, y\}$  triplets.

#### 3.3 Neural Network Training

The training processes for the multilayer perceptron and the functional link network were performed off-line, i.e. not simultaneously with the wing deflection simulation. The acquisition of FLN weights  $w$  for all 1000 wing deflections required the training of the FLN 1000 times and saving the resulting  $w$ 's.

The output and hidden layer of the MLP were trained for 5000 iterations using the backpropagation algorithm. A decreasing learning rate and momentum term [7] was used,

starting at 0.3 and 0.4 respectively. The mean squared error at the output of less than 1% was accomplished by the MLP. The FLN training was also very successful. The authors applied the FLN to robot control in a previous project [2] and could build on that experience. The network was trained for 20000 iterations using the backpropagation algorithm. A decreasing learning rate and a increasing momentum term was used, starting at 0.3 and 0.1 respectively, and a mean squared error of less than 1% was accomplished by the FLN. The training data for both networks were normalized to the specified input and output range of each network.

## 4 Neural Network Processor

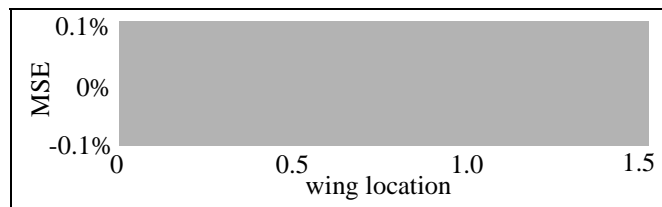
Real-time performance is a stringent requirement for all controllers. For the present modeling problem real-time operation is achieved by implementing the trained MLP and the FLN on a specialized multiple-instruction multiple-data (MIMD) neural network processor (NNP). The processor provides the high throughput required for on-line neural network processing for complex tasks. The NNP has been developed specifically for neural network applications by Accurate Automation Corporation [8]. The processor is capable of implementing 8K neurons with 32K interconnections and computes 140 million connections per second (CPS) at 35 MHz clock frequency. Up to 8 NNPs can be stacked and operate together, providing 1 billion CPS. The multilayer perceptron and the functional link network used for the present study have been implemented on the NNP. The assembler code on custom-made hardware demonstrated real-time capability as required for high performance aircraft, where time issues are critical.

## 5 Simulation Results

The proof-of-concept configuration of the neural wing deflection model presented here gave accurate results. Figure 4 shows the mean squared error (MSE) of a vertical wing deflections as a function of the wing position ( $x$ -axis). The original wing size and deflection was normalized to the appropriate range of the neural networks to prevent saturation of the sigmoid transfer functions. This normalization is a simple linear transform and is easily to invert at the network output, back to the real data ranges. Under the idealized conditions assumed for the present work the MSE of the modeled deflection was less than 1%. This successful simulation of a flexible aircraft wing demonstrates the potential of artificial neural networks for the geometrical approximation of flexible structures.

Further research must investigate the influence of sensor noise, the order of the FLN, and the minimum number of sensors which provides sufficient approximation accuracy.

Furthermore, the potential of multi-dimensional approximation of surfaces and the associated neural network architectures will be considered. The use of signed curvature values will also be investigated to guarantee the acquisition of unique training sets for neural network learning.



**Figure 5.** Mean squared error in % for a set of input-output data.

## 6 References

- [1] S.J. Brattke, "Distributed Adaptive Control of Flexible Structures", unpub. notes, Univ. Massach.. Amherst, Comput. Science Dept., 1993.
- [2] C. Cox, J. Edwards, R. Saeks, R. Pap, K. Mathia, "Adaptive Semi-Autonomous Robotic Neurocontroller", *Proc. SPIE, Appl. Artif. Neural Networks V*, pp. 440-445, 1994.
- [3] T.L. Hemminger, Y.H. Pao, "Detection and Classification of Underwater Acoustic Transients using Neural Networks", *IEEE Trans. Neural Networks*, Vol. 5, No. 5, pp. 712-718, 1994.
- [4] Lockheed Fort Worth Corporation, "Vital Advanced Inertial Networks PRDA", Technical Report.
- [5] A. Makroglou, R.K. Miller, "Computational Results for a Feedback Control for a Rotating Viscoelastic Beam," *J. Guid., Control, and Dyn.*, Vol. 17, No. 1, 1994.
- [6] Y.H. Pao, *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, MA, 1989.
- [7] D.E. Rumelhart, J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, Ch. 8, MIT Press, Cambridge, MA, 1986.
- [8] R. Saeks, K. Priddy, "On the Design of an MIMD Neural Network Processor," *Proc. World Congress Neural Networks '95*, San Diego, Vol. II, pp. 590-595.
- [9] G.B. Thomas, R.L. Finney, *Calculus and Analytical Geometry*, Addison-Wesley, Reading, MA, 1990.
- [10] D.F. Specht, "Probabilistic Neural Networks and the Polynomial Adaline as Complementary Techniques for Classification", *IEEE Trans. Neural Networks*, Vol. 1, No. 1, pp. 111-121, 1990.
- [11] G.G. Yen, "Reconfigurable Learning Control in Large Space Structures," *IEEE Trans. Control Systems Technology*, Vol. 2, No. 4, pp. 362-370, 1994.
- [12] S. Yurkovich and A.P. Tzes, "Experiments in Identification and Control of Flexible-Link Manipulators," *IEEE Contr. Sys. Mag.*, Vol. 10, No. 2, pp. 41-47, 1990.