A NEURAL NETWORK SATELLITE ATTITUDE CONTROLLER WITH ERROR BASED REFERENCE TRAJECTORY

Valdemir Carrara[+]
Atair Rios Neto[++]
[+] Instituto Nacional de Pesquisas Espaciais – INPE/MCT
CEP 12201-970 CP 515 São José dos Campos, SP - Brazil
E-mail: val@dem.inpe.br
[++] Instituto de Pesquisa e Desenvolvimento, Universidade do Vale do Paraíba.
CEP 12245-720 São José dos Campos, SP – Brazil
E-mail: atair@univap.br

Abstract

Feedforward neural networks are investigated in this work to verify its ability to control the attitude of a satellite. Neural nets are a promising tool for attitude control due to its inherent nonlinear behavior, which makes them a natural candidate to control nonlinear systems. Nevertheless, as will be shown, to obtain a neural 3 axis attitude control is not as simple as a conventional SISO net control. Main difficulties are the large amount of training data, in order to assure the complete understanding of the attitude dynamics by the neural net, and also the fact that attitude control is a MIMO instead of a SISO system. Regardless of these draw backs, some results concerning attitude control will be shown.

Introduction

The training process of a control neural network depends on the condition of the dynamic system having at least a local inverse around a reference trajectory, and is very much affected by dynamic system complexity. Thus for a given training method, the learning process can or can not converge, depending on system dynamics[1]. Several methods (generalized, specialized, predictive control) to obtain the inverse dynamic model have been established to guarantee training convergence[1-2], and some particular features make them more or less appropriate depending on the application. These training methods use a feedforward reference trajectory as an input to the neural net controller. This arrangement is suggested based on the way humans control their movements and normally applies to robotic systems. However, for several applications, it is more important to correct residual errors than to follow an arbitrary trajectory. This is certainly the case of a satellite attitude control, where small but effective disturbances deviates the target pointing and where highly nonlinear dynamics also makes attitude maneuvering sometimes a difficult process. In this case, a static input reference trajectory in a feedforward neural net controller can not provide the necessary dynamic information in order to compensate for the attitude deviations. For systems that need a closed loop control, an error based reference trajectory is suggested. The error signal has the characteristic of generating a null control when the error drops to zero, allied to enough information on magnitude and velocity. The neural net, on the other hand, can theoretically learn the nonlinear behavior of the satellite, resulting in a nonlinear feedback error control. The proposed scheme is similar to the inverse generalized method[1], with the difference that the system is now controlled in closed loop. The number of neurons in the network and the number of training points were adjusted interactively, assuring minimum learning time. A simulation presenting the neural control is compared to a conventional PD controller. Although the neural control is still far behind the PD, it is expected that with specific goal-directed training methods the nonlinear characteristic of the neural net can be better utilized.

Keywords: Neural Network Controller, Attitude Control, Feedback Control.

Neural Networks

Artificial neural nets (ANN) are composed of individual processing units called neurons grouped in layers. In feedforward nets each neuron applies an activation function *f* to the sum of the weighted outputs of the previous layer. For hidden layers, the activation function generally is a biased nonlinear differentiable function like the sigmoid or hyperbolic tangent, for instance, while the output layer can be a linear function[1]. A supervised training method adjusts the neuron weights based on error obtained at the net output and applying some optimization rule. Training consists of an interactive process in which the weights are adjusted by propagating the output error through the network layers. Nonlinear continuous functions can be

approximated with a given accuracy by a 2 layer neural net with linear function in the output and the sigmoid activation function:

$$f(x) = \frac{1 - e^{-x}}{1 + e^{x}} \tag{1}$$

in the hidden layer[3-4]. A feedforward network composed by l layers, as shown in Figure 1, can be seen as a mapping function with n_0 input elements and n_l output parameters. If x_i^k is the output of the i^{th} neuron of layer k, w_{ij}^k is the weight of the j^{th} input (coming from the j^{th} neuron of the preceding layer) and f^k is the activation function of layer k, then:

$$x_i^k = f^k\left(\bar{x}_i^k + b_i^k\right) = f^k\left(\sum_{j=1}^{n_{k-1}} w_{ij}^k \, x_j^{k-1} + b_i^k\right) \tag{2}$$

where b_i^k is the neuron bias that allows the neuron to present a non-null output for a null input.

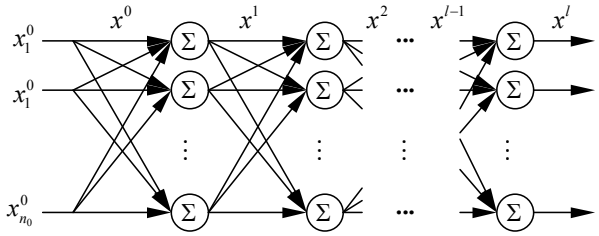


Fig. 1 A feedforward neural network

Generally the neuron bias can be obtained together with the weights, by assuming the inclusion of new unit input. In a vector-matrix representation form the preceding equation yields

$$x^k = f^k\left(\bar{x}^k\right) = f^k\left(W^k x^{k-1}\right) \tag{3}$$

where the weight matrix of layer k, W^k, includes the neuron bias:

$$W^k = \begin{bmatrix} w_{11}^k & \cdots & w_{1n_{k-1}}^k & b_1^k \\ w_{21}^k & \cdots & w_{2n_{k-1}}^k & b_2^k \\ \vdots & & \vdots & \vdots \\ w_{n_k 1}^k & \cdots & w_{n_k n_{k-1}}^k & b_{n_k 1}^k \end{bmatrix}. \tag{4}$$

The dimensions of the output vector x^k and the weight matrix W^k are now n_k+1 and $n_k \times n_{k-1}$+1, respectively.

The increasing number of hidden layers normally makes the neural net to better represent the dynamical system and to reduce the output error[5-6], even taking the same number of neurons. Nevertheless, the capacity of generalization, i. e. the ability to interpolate between points where the neural net was not trained is more accentuated on nets with few or even only one hidden layer[7]. The number of neurons in the hidden layers is important for the approximation degree: few neurons tend to decrease the stability and result bad approximation, too much neurons cause oscillation on the output between the trained points[8].

Backpropagation algorithm

Neural nets have three major advantages when compared with traditional function approximation methods. First is parallel processing structure, which allows quickly response in parallel computers. Second, they are able to handle with large number of input elements, and do not need filtering or state estimation processing. Third, the weights are easily obtained by using training procedures, that gradually teaches the net how to respond to a given input. The training process normally minimizes the output error through the application of an optimization method. These methods need to know with some extent how the net output varies with respect to a given neuron weight. This can be achieved with the back propagation algorithm developed by Werbos[1], which obtains the partial derivative of the output elements in a recursive way. In matrix form the back propagation algorithm the derivative of the output vector with respect to the j^{th} weight of the i^{th} neuron of the k^{th} layer results the expression:

$$\frac{\partial x^l}{\partial w_{ij}^k} = \Delta^k \begin{bmatrix} 0 \\ \vdots \\ x_j^{k-1} \\ \vdots \\ 0 \end{bmatrix}, \tag{5}$$

where Δ^k is the back propagation matrix, obtained from:

$$\Delta^k = \Delta^{k+1} W^{k+1} F^k \tag{6}$$

with initial condition $\Delta^l = F^l$, where F^k is a diagonal matrix with the derivatives of the activation function f^k:

$$F^k = \frac{df^k(\bar{x}^k)}{d\bar{x}^k} = \begin{bmatrix} f^{k\prime}(\bar{x}_1^k) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & f^{k\prime}(\bar{x}_{n_k}^k) \end{bmatrix} \quad \textbf{(7)}$$

It should be noted that, due to the inclusion of the neuron bias on the weight matrix, F^k should be a $n_{k-1}+1 \times n_{k-1}+1$ matrix, with the last diagonal element equal to zero. However, to reduce computation both F^k and W^k can be resized with elimination of the last row when performing matrix products.

Steepest descent or gradient method is today the most common training procedure. It is ease to implement in computers, is very fast but converges in a strongly slow rate. This certainly is the main reason of the extremely long training times in most neural net applications. Nevertheless, there are other training methods that show improved learning speed, as the least square[6-9], or the Levenberg-Marquardt algorithm[10-11]. In spite of the training procedure, the network weights can be updated at each input presentation, in a so-called adaptive training, or at the end of a complete set of input data, known as batch training.

Adaptive training allows the network to learn the system dynamics in real time, although the learning can also be done offline. If the learning rate is too large or if the system remains at a specific state for long time, the network adjust the weights to the last trained position and the learning remains incomplete. The same is true if the system do not pass to some points or regions in the state space. Batch training appears to avoid such underfitting in the first case, but depending on the system complexity, the training can or can not be performed over the entire state space. The training of a network attitude controller can not be done after in-orbit injection, because during the learning process the erroneous control can put the satellite in a dangerous situation. So, computer simulated dynamics shall be used for training in order to guarantee the controllability before launching. This procedure allows also train the neural net over the entire state space, and not only in one particular trajectory. Nevertheless, some problems arise from this solution, mainly due to the large number of training points necessary to inform the system dynamics to the neural net. For example, for a 3 degrees of freedom second order system (like a satellite attitude), the neural input can be determined with 9 variables (3 for each position, velocity and control). If one admits 5 samples for each variable, then the training set will have 5^9 = 1,953,125 training points in order to cover the state space, which is almost impossible to get with the computer and memory available today. The training process could take several months and the resulting net would be so large that real-time application would turn to be a mere desire. Fortunately, at least in theory, the training set do not need to be so large, by taking into consideration the possibility that the network can acquire enough system information by generalizing and interpolating the input data. In this sense, a statistical method, similar to the Monte Carlo, can be applied by generating random points in the state space. The problem now will be to find the necessary number of points and the corresponding number of neurons that learns the dynamic behavior.

Training approaches

One of the most peculiar aspects in neural net control is how to obtain the control signal. Hunt[1] suggests some well know training approaches, like the generalized inverse, the indirect and the specialized inverse methods. Each one of these models has advantages and also disadvantages concerning the quality of results. Generalized inverse learning presents some negative properties when the training is performed with the real system, as there is no guarantee that the system output covers totally the state space. Of course this problem will not happen in case of a numeric simulated system. Indirect model presents some instability during training, depending on the system dynamics. Specialized inverse method requires a network direct model in order to establish a relation between the direct model output error and the control network output error. In a simulated system, nevertheless, specialized and generalized methods are equivalents and so the results presented here were obtained with the generalized inverse model, as shown in Figure 2.

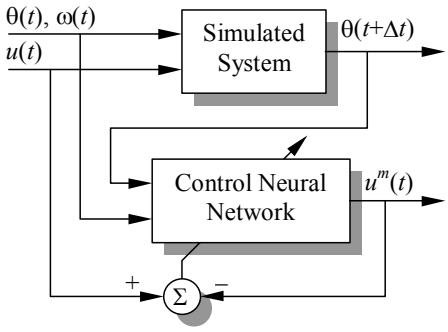


Fig. 2 – Control network training model.

Inputs to the control network are the state (3 attitude angles and 3 angular velocities) at time t and the propagated position at time $t+\Delta t$. The output is the control signal (torque) u(t). System dynamics is simulated considering a non-perturbed 3 axis rigid body, with inertia equals to 23, 23 and 11 kg.m^2. Torque is

provided by gas jets, supposed regulated by a PWM device. Maximum available torque is considered u_{max} = 1.5 Nm. The network is trained with the Levenberg-Marquardt method[10-11].

The network final error depends on the number of hidden neurons and the number of training points. In order to achieve a given precision, the larger the training set, the greater the number of hidden neurons. If u_{max} is the maximum available torque, then the network error shall be a fraction of this value, 1% for instance. This value, nevertheless, has to be selected carefully, as the number of hidden neurons (and also the training time) increases quickly with the required net accuracy. Unfortunately, there is no theory that gives the number of neurons as function of the number of training points. So it is adopted an iterative approach: the training is performed first with a small network with few training points. If the required accuracy is not met after a predefined number of training iterections, then the number of neurons is increased by a factor larger than 1 (1.4 in this case). On the other hand, if the training succeed, then a new random input set is generated and a new output mean error is calculated. If the new error is still below the required accuracy, the training process is completed. Otherwise the number of training points is multiplied by a factor greater than one (1.4) and the whole process is repeated. With such algorithm the number of training points and hidden neurons grows up to the minimum necessary to learn the system dynamics.

Training was carried out by generating random positions between ± 20° and ± 1 rpm on each axis. Starting with 8 neurons and 256 training points, the required output error was reached with 24 neurons, and the learning process succeed with only 2576 points, as shown in Figure 3.

Once trained, the ANN controller is used in conjunction with the attitude simulator in order to validate the control. Nevertheless, as shown in Figure 2, the network needs to know previously the attitude at time $t+\Delta t$. This information, not available at current time, can be replaced by the reference trajectory $\theta^r(t+\Delta t)$, as suggested by Hunt[1] and depicted in Figure 4. This procedure however characterizes a feedforward controlled system, as the network does not correct the output error of the system. In order to feed back the attitude error, it is adopted a reference trajectory calculated as function of the error between the target and the actual attitudes. The reference trajectory can, for instance, be proportional to the error:

$$\theta^r(t+\Delta t) = \theta(t) - a_p\,[\theta(t) - \theta^t(t)] \qquad \textbf{(8)}$$

where $\theta^t(t)$ is the target attitude. This procedure led the system to the target position as expected, but with increasing oscillation as the satellite has no way to dump the angular motion. So it is necessary to include a dump factor on the reference trajectory proportional to the angular velocity. If the target attitude is a null attitude, then θ^r becomes:

$$\theta^r(t+\Delta t) = \theta(t) - a_p\,\theta(t) + a_d\,\omega(t)\,. \qquad \textbf{(9)}$$

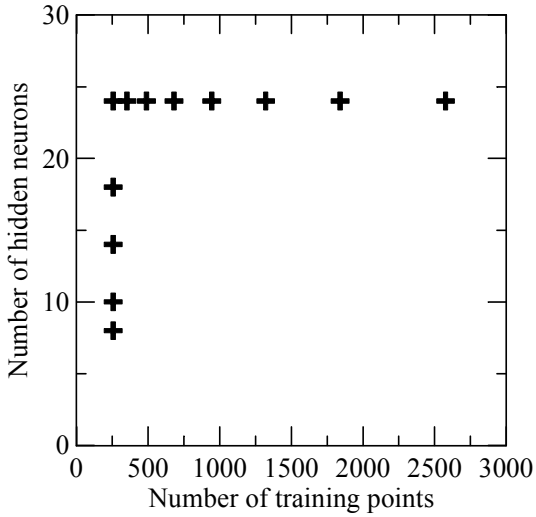


Fig. 3 – Number of hidden neurons as function of the number of training points.

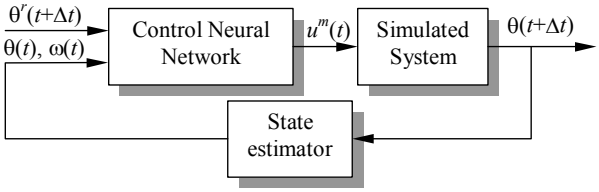


Fig. 4 – Neural Network Control

The stability of the ANN control revealed a difficult task, as the scalars a_p and a_d had to be adjusted by trial and error, and the stability range is very tight. Small differences in a_p and a_d move from long time convergence to instability. Considering a_p = 0.08 and a_d = 1.2, the result for a 200 seconds attitude simulation is shown in Figure 5. As can be seen, attitude oscillation could not be removed entirely. Maybe this can be done by using different gains for individual satellite axis instead of only one value for all 3 axes. The ANN also presented a small but non-zero bias that causes the control to respond even when the target has reached. This made the trajectory to dump slowly near the final attitude. It is possible also that the nonlinear behavior of the network caused the oscillations, but this can be

proved only increasing the number of hidden neurons and training points. Initial conditions for curves in Figure 5 are: attitude angles in pitch, roll and yaw, θ = (10°, 15°, -5°) and angular velocity ω = (0.1, 0.6, 0.2) rpm.

Note that the reference trajectory, as proposed in Equation 9 is similar to a PD (proportional and derivative) controller. The difference is that the torque obtained by the network is based on a nonlinear attitude dynamics, whereas the PD normally obtains the gains upon linearized assumptions. In fact, the performance of PD controller is better than the ANN, as seen in Figure 6, even considering that the PD gains were not optimized. Both curves have identical initial and steady state conditions. Gains for the PD controller are proportional ap= 0.5 and derivative ad= 7.5. Note also that the PD reaches the target in 20 seconds, whereas the ANN controller takes more than 200 seconds.

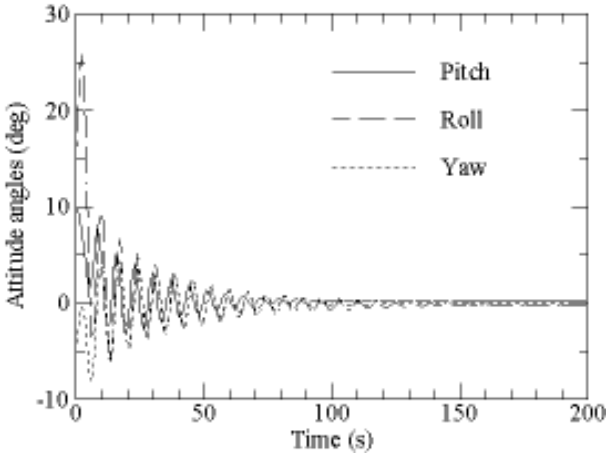


Fig. 5 – Attitude simulation with a ANN controller

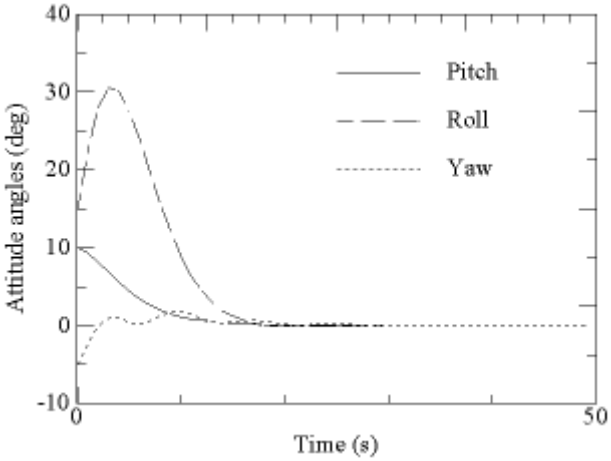


Fig. 6 – Attitude simulation with a PD controller

Conclusions

This paper presented a comparison of an ANN attitude and a conventional PD controllers. ANN acting as a nonlinear system control has some intrinsic advantages that can be exploited in further studies. Nevertheless, if by one side there is not still a global theory for nonlinear control, on the other hand the large number of the ANN parameters that need to be adjusted by trial and error makes the training process an exhaustive task. It is important to note also that there is not too many examples, in literature, concerning system control with several degrees of freedom by means of ANN. As became clear the larger the number of state variables, the bigger the network. In fact, the first attempt to train the ANN was carried out with maximum torque of u_{max} = 0.15 Nm, more realistic with respect to the satellite size. However, as the output accuracy was a fraction of this value (1% indeed), training was not completed even with 128 hidden neurons. Training times was so large that the process had to be stopped.

References

[1] Hunt, K. J.; Sbarbaro, D.; Zbikowski, R.; Gawthrop, P. J. Neural networks for control systems - a survey. *Automatica*, v. 28, n. 6, p. 1083-1112, 1992.

[2] Kawato, M; Uno, Y.; Isobe, M; Susuki, R. Hierarchical neural network model for voluntary movement with application to robotics. *IEEE Control Systems Magazine*, v. 8, n. 2, p. 8-15, Apr. 1988.

[3] Cybenko, G. Approximation by superposition of a sigmoidal function. *Mathematics of Controls, Signals ans Systems*. v. 2, n. 4, p. 303-314, 1989.

[4] Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, v. 2, n. 5, p. 359-366, 1989.

[5] Nguyen, D. H.; Widrow, B. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, v. 10, n. 3, p.18-23, Apr. 1990.

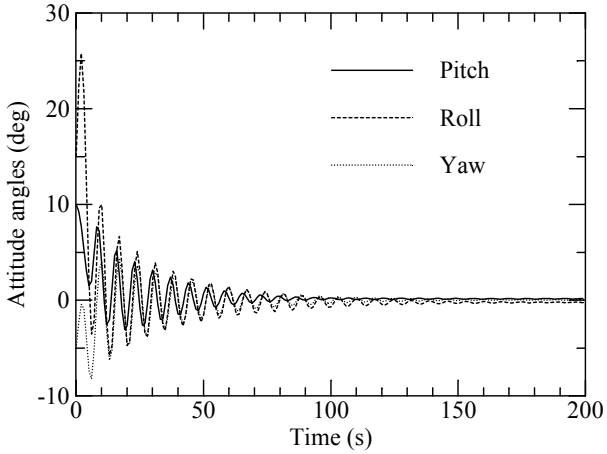
[6] Chen, S.; Billings, S. A. Neural networks for nonlinear dynamic system modelling and identification. *International Journal of Control*, v. 56, n. 2, p. 319-346, 1992.

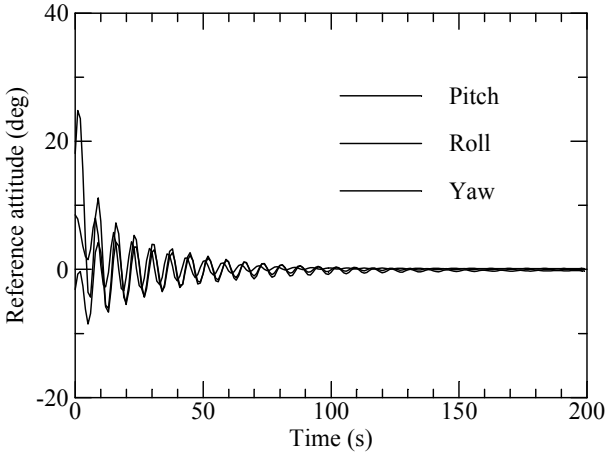
[7] Baffes, P. T.; Shelton, R. O.; Phillips, T. A. *NETS, a neural network development tool*. Huston, Lyndon B. Johnson Space Center, 1991. (JSC-23366)

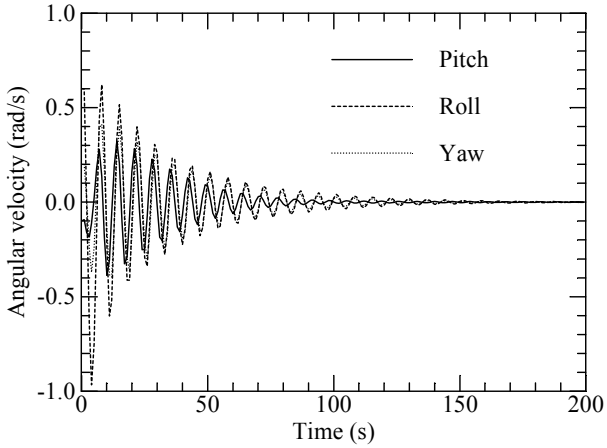
[8] Billings, S. A.; Jamaluddin, H. B.; Chen, S. Properties of neural networks with applications to modelling non-linear dynamical systems. *International Journal of Control*. v. 55, n. 1, p. 193-224, 1992.

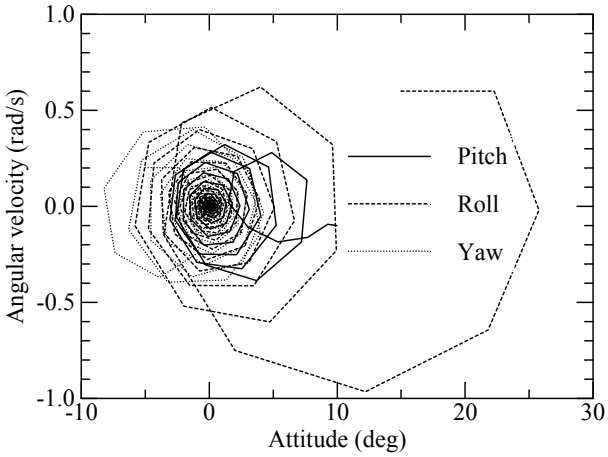
[9] Carrara, V.; Varotto, S. E. C.; Rios Neto, A. Satellite Attitude Control Using Multilayer Perceptron Neural

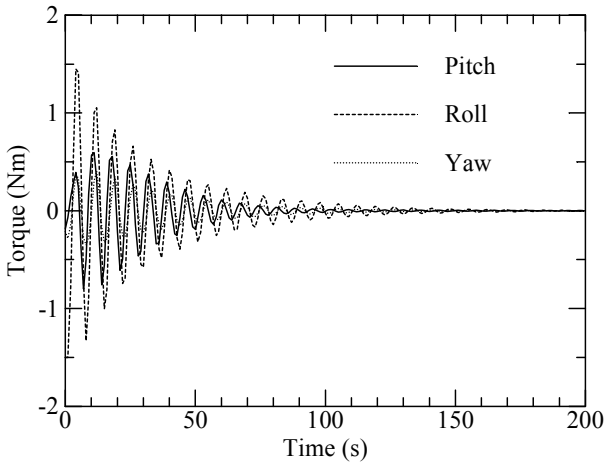
Networks (98-345). *Advances in the Astronautical Sciences*. Vol. 100, Part 1, 1998. p. 565-579.
[10] Hagan, M. T.; Menhaj, M. Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, V. 5, n. 6, pp. 989-993, 1994.
[11] Demuth, H.; Beale, M. Neural Network Toolbox User's Guide Version 3.0. MathWorks, 1997 (in PDF file).

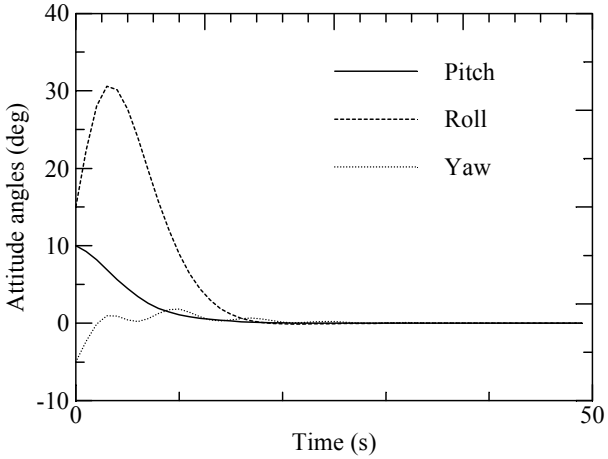

Attitude angles (deg)
Time (s)
Pitch
Roll
Yaw


Reference attitude (deg)
Time (s)
Pitch
Roll
Yaw


Angular velocity (rad/s)
Time (s)
Pitch
Roll
Yaw


Angular velocity (rad/s)
Attitude (deg)
Pitch
Roll
Yaw


Torque (Nm)
Time (s)
Pitch
Roll
Yaw


Attitude angles (deg)
Time (s)
Pitch
Roll
Yaw

controle PD