Chapter 9

Control of Rolling in Ships by Means of Active Fins Governed by a Fuzzy-Logic Controller

The paper, "Neural-Network and Fuzzy-Logic Control to Suppress the Rolling Motion in Ships by Means of Active Fins," (reference [55]), by Liut *et al.*, is based on the contents of this chapter.

9.1 Introduction

In this chapter, we extend many results of our neural-network-control research to fuzzy-logic control. To this end, we revisit the problem of roll reduction by means of active stabilizers discussed in Chapter 5. The hydrodynamics of the problem is the same as what was discussed in Chapter 2, but here we introduce a fuzzy-logic controller to govern the fin motions in order to reduce the rolling motion.

We undertake our discussion with two objectives in mind: 1) to describe the basis of fuzzy-logic control used to command the fin deflections; 2) to extend the moment-matching training procedure, introduced in Chapter 5, to a fuzzy-logic controller.

In this chapter, we show how a neural-network controller and a fuzzy-logic controller can be treated in a similar manner though the principles that inspired their development were different. This does not come as a surprise since, as it was pointed out in Chapter 1,

fuzzy-logic systems can be seen within the perspective of radial-basis-function neural networks.

9.2 Fuzzy-Logic Control

A fuzzy-logic controller associates different sets of input data to a system with the desired control actions for the system. Each one of such associations defines a rule. The idea behind fuzzy logic is to convert crisp or *quantitative* input data into appropriate fuzzy or *qualitative* data that can be handled logically in order to produce a desired output. This output has an abstract or qualitative structure, and through an appropriate defuzzification scheme, crisp output data can be obtained. The inspiring idea behind this method was trying to reproduce some processes of abstract logical thinking. Such logical processes are simulated by rules, which associate different input data with a desired output. An example of a simplified rule for the fuzzy-logic controller described in this chapter might be expressed as:

1) If the rolling angle is small and positive
2) and the pitching angle is small and positive
3) and the rolling rate is medium and negative

- **then** the antysimmetric deflection of the fins must be small and positive

In real-life applications, we do not operate with *small* or *medium* values, but with numerical data. Therefore, a fuzzification process must be implemented to convert these quantitative, numerical or *crisp* data into qualitative, logical or *fuzzy* data. The word *fuzzy* may be misleading if it is interpreted as an approximate or vague procedure. This approach has the approximation any mathematically based procedure can have and is anything but vague. After the logical operations are completed, a defuzzification scheme

is used to convert the logical results; for example *the antysimmetric deflection of the fins must be small and positive,* should be given in a numerical way, as for example, $+1.534^{o}$ angle of deflection for the port fin, and -1.534^{o} angle of deflection for the starboard fin.

A classical way to fuzzify the input/output data is to define the so-called membership functions. Each membership function associates the *degree of belonging* of any crisp value of some variable to an equivalent qualitative or *fuzzy* value. For example, if we want to determine the *degree of smallness* of the value 1.2^{o} of rolling angle, a $M_{small}(\phi)$ membership function may be defined to assess this. If $M_{small}(1.2^{o})$ yields a relatively large value, then 1.2^{o} can be considered as a small rolling angle. If $M_{small}(1.2^{o})$ yields a relatively small value then 1.2^{o} might not be considered a small rolling angle. The membership functions are scaled according to an arbitrary convention. In most applications, they are scaled from zero to one. If we apply this convention to our example, then if $\phi = 1.2 \rightarrow M_{small}(1.2) = 0.9$ then ϕ=1.2 is predominately a small rolling angle (since 0.9 is close to one), whereas if $\phi = 1.2 \rightarrow M_{small}(1.2) = 0.1$ then ϕ=1.2 would not be predominately a small rolling angle. Membership functions can have different forms. For this work, we use Gaussian functions and singletons. In Figure 9-1 a set of Gaussian membership functions is defined for a certain input vector **x** of dimension *n*. This input vector is associated with a scalar function *y* through a set of fuzzy-logic rules, as described below. The output of the fuzzy-logic system gives the computed value of *y* through a set of output membership functions defined by singletons. If the same number of membership functions *R* is used for each element of **x** and the function *y*, a matrix of membership functions **M** can be defined according to:

$$\mathbf{M} = \begin{bmatrix} \mathrm{M}\, x_{11} & \mathrm{M}\, x_{12} & \mathrm{M}\, x_{13} & \cdots & \mathrm{M}\, x_{1R} \\ \mathrm{M}\, x_{21} & \mathrm{M}\, x_{22} & \mathrm{M}\, x_{23} & \cdots & \mathrm{M}\, x_{2R} \\ \vdots & \vdots & \vdots & & \vdots \\ \mathrm{M}\, x_{n1} & \mathrm{M}\, x_{n2} & \mathrm{M}\, x_{n3} & \cdots & \mathrm{M}\, x_{nR} \\ \mathrm{M}\, y_1 & \mathrm{M}\, y_2 & \mathrm{M}\, y_3 & \cdots & \mathrm{M}\, y_R \end{bmatrix} \tag{9.1}$$

By arranging the membership functions in this way, we can make each column of **M** define a rule, where $\mathrm{M}\, x_{1r} \cdots \mathrm{M}\, x_{nr}$ are the *antecedents* or *premises* of each rule, and $\mathrm{M}\, y_r$ are the *consequences* or *conclusions* of each rule, for $r = 1 \cdots R$.

An efficient way to operate logically with these rules, and at the same time perform the fuzzifying/defuzzifying procedure is to apply Sugeno's zero-order fuzzy-inference system. This is the procedure used by ANFIS's [90, 91, 92]. For our problem this is given by,

$$y = \frac{\sum_{r=1}^{R} y_r \left[\prod_{i=1}^{n} \mathrm{Exp}\left(\frac{x_i - c_{ir}}{\sigma_{ir}} \right)^2 \right]}{\sum_{r=1}^{R} \left[\prod_{i=1}^{n} \mathrm{Exp}\left(\frac{x_i - c_{ir}}{\sigma_{ir}} \right)^2 \right]} \tag{9.2}$$

where the c_{ir} are the centers of the membership functions for each variable x_i of the antecedent (input data) **x**, and the σ_{ir} are the corresponding spread factors of the Gaussian functions. The membership functions used for the consequences are a set of singletons y_r (for $r = 1 \cdots R$). Figures 9.2 and 9.3 illustrate this. Equation (9.2) is also

referred to as a fuzzy-logic Universal Approximator, and it approximately simulates the following logical, fuzzifying/defuzzyfying operation:

$$y = \underset{r}{\text{Max}}\left[y_r \ \underset{i}{\text{Min}}\left(\text{M}\, x_{ir}(x_i)\right)\Big|_{\substack{\text{for each}\\ \text{rule } r}} \right] \qquad \begin{array}{l}\text{for} \quad i = 1 \cdots n \\ \\ \text{for} \quad r = 1 \cdots R\end{array} \tag{9.3}$$

One reason for using a Sugeno approximator in this work is that it makes it possible to implement the backpropagation technique for the training process we describe in the next section.

The input data to the fuzzy-logic system, **x**, take into account most of the state variables of the system, the rudder deflection and some information about the neighboring sea:

$$\mathbf{x} = \left[Y_0, \quad Z_0, \quad v_x, \quad v_y, \quad v_z, \quad \phi, \quad \theta, \quad \psi, \quad \omega_x, \quad \omega_y, \quad \omega_z, \quad \delta_r, \quad w_1, \cdots w_h, \quad b \right]^T \tag{9.4}$$

The elements $w_1, \cdots w_h$ of vector **x** are data related to the neighboring sea as was explained in Chapter 5.

9.3 The Training Procedure

In this section, we describe how to train the fuzzy-logic controller. First, we extend the backpropagation algorithm described in Chapter 4 for neural-network systems to fuzzy-logic systems. Then we describe how the same moment-matching procedure discussed for the neural-network controller of Chapter 5 can be used to provide an effective training error signal for the fuzzy-logic controller. Finally, we explain how to initialize

the fuzzy-logic controller combining a clustering algorithm with the moment-matching procedure.

9.3.1 The Backpropagation Algorithm

In this subsection, we show how the same backpropagation concepts described in Chapter 4 for neural networks can be extended to fuzzy-logic systems. We start our explanation by defining an error $e = e(t)$ as the difference between a more desirable deflection of the fin $d(t) = \delta_d(t)$ and the deflection of the fin commanded by the controller, $y(t) = \delta_c(t)$. Later we explain how to relate this difference to the desired motion. We can write this error as

$$e(t) = \delta_d(t) - \delta_c(t) \tag{9.5}$$

To abbreviate the notation the time argument is omitted from all variables in the following discussion.

A measure of the performance Θ is defined as half of the square of the error:

$$\Theta = \frac{1}{2}e^2 = \frac{1}{2}(d - y)^2 \tag{9.6}$$

The backpropagation technique uses this performance Θ as a reference for the adjustments that must be performed to the center c_{ir} of each membership function, their spreads σ_{ir} and the magnitude of the singletons y_r. The first goal is to compute the derivatives of Θ with respect to y_r, c_{ir}, and σ_{ir}. This is done at every time step. These derivatives are then used to adjust y_r, c_{ir}, and σ_{ir} (at every time step). We can visualize

the fuzzy-logic system in Figure 9-4. To make the derivation process easier to follow we define the following functions (see Figure 9-4):

$$v_{ir} = \frac{x_i - c_{ir}}{\sigma_{ir}} \tag{9.7}$$

$$g_{ir} = \mathrm{Exp}\left(-v_{ir}^2\right) \tag{9.8}$$

$$z_r = \prod_{i=1}^{n} [g_{ir}] \tag{9.9}$$

$$a = \sum_{r=1}^{R} [y_r \ z_r] \tag{9.10}$$

$$b = \sum_{r=1}^{R} [z_r] \tag{9.11}$$

$$y = \frac{a}{b} = \frac{\sum_{r=1}^{R} [y_r z_r]}{\sum_{r=1}^{R} [z_r]} \tag{9.12}$$

The derivatives of Θ with respect to y_r can be computed as

$$\frac{\partial \Theta}{\partial y_r} = \frac{\partial \Theta}{\partial e}\frac{\partial e}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial y_r} \qquad \text{for} \qquad r = 1 \cdots R \tag{9.13}$$

where if we use the definitions given by equations (9.7)-(9.12) we get:

$$\frac{\partial \Theta}{\partial y_r} = (e)(-1)\left(\frac{1}{b}\right)(z_r) = -\frac{e\, z_r}{b} \qquad \text{for} \qquad r = 1 \cdots R \tag{9.14}$$

The derivatives of Θ with respect to c_{ir} can be computed as

$$\frac{\partial \Theta}{\partial c_{ir}} = \frac{\partial \Theta}{\partial e}\frac{\partial e}{\partial y}\frac{\partial y}{\partial z_r}\frac{\partial z_r}{\partial v_{ir}}\frac{\partial v_{ir}}{\partial c_{ir}} \qquad \begin{array}{l} \text{for} \quad i = 1 \cdots n \\ \\ \text{for} \quad r = 1 \cdots R \end{array} \tag{9.15}$$

where if we use the definitions given by equations (9.7)-(9.12) we get:

$$\frac{\partial \Theta}{\partial c_{ir}} = (e)(-1)\left(\frac{1}{b} y_r - \frac{a}{b^2} 1\right)[z_r(-2\, v_{ir})]\left(\frac{1}{\sigma_{ir}}\right) \qquad \begin{array}{l} \text{for} \quad i = 1 \cdots n \\ \\ \text{for} \quad r = 1 \cdots R \end{array} \tag{9.16}$$

Rearranging terms, we get:

$$\frac{\partial \Theta}{\partial c_{ir}} = \frac{2\, e\,(y_r - y)\, z_r\,(x_i - c_{ir})}{b\, \sigma_{ir}^2} \qquad \begin{array}{l} \text{for} \quad i = 1 \cdots n \\ \\ \text{for} \quad r = 1 \cdots R \end{array} \tag{9.17}$$

To obtain $\frac{\partial \Theta}{\partial \sigma_{ir}}$ we proceed analogously as it was done for $\frac{\partial \Theta}{\partial c_{ir}}$, and we get:

$$\frac{\partial \Theta}{\partial \sigma_{ir}} = \frac{2\, e\,(y_r - y)\, z_r\,(x_i - c_{ir})^2}{b\, \sigma_{ir}^3} \qquad \begin{array}{l} \text{for} \quad i = 1 \cdots n \\ \\ \text{for} \quad r = 1 \cdots R \end{array} \tag{9.18}$$

Based on the Hebbian Principle, (Hebb [29]), the derivatives of Θ with respect to y_r, c_{ir}, and σ_{ir} obtained in equations (9.14), (9.17) and (9.18) at every time step are made proportional to the updates for y_r, c_{ir}, and σ_{ir}, as follows:

$$y_r(k+1) = y_r(k) + \lambda_y \frac{\partial \Theta}{\partial y_r}(k) \qquad \text{for} \quad r = 1 \cdots R \tag{9.19}$$

$$c_{ir}(k+1) = c_{ir}(k) + \lambda_c \frac{\partial \Theta}{\partial c_{ir}}(k) \qquad \begin{array}{l} \text{for} \quad i = 1 \cdots n \\ \text{for} \quad r = 1 \cdots R \end{array} \tag{9.20}$$

$$\sigma_{ir}(k+1) = \sigma_{ir}(k) + \lambda_\sigma \frac{\partial \Theta}{\partial \sigma_{ir}}(k) \qquad \begin{array}{l} \text{for} \quad i = 1 \cdots n \\ \text{for} \quad r = 1 \cdots R \end{array} \tag{9.21}$$

where k stands for the current time step, and where λ_y , λ_c and λ_σ are constants commonly referred to as the (fuzzy-logic) learning parameters for y_r, c_{ir}, and $\sigma_{ir,}$ respectively. They are determined heuristically.

9.3.2 The Moment-Matching Procedure

The same moment-matching procedure described in Chapter 5 for a neural network controller is now applied to the fuzzy-logic controller. As before, the present strategy is to reduce $|\dot{\omega}_x|$ as much as possible. To attain this, small adjustments are introduced to y_r, c_{ir}, and σ_{ir}. As shown in Chapter 5, equation (5.22), a time-step-by-time-step adjustment is introduced in order to minimize the rolling motion as follows:

$$\Delta\mu_x = -\tilde{\lambda}\, I_{xx}\dot{\omega}_x \cong \gamma\left(\delta_d - \delta_c\right) \tag{9.22}$$

where $\Delta\mu_x$ is a change in the rolling moment triggered by the change in the fuzzy-logic parameters, γ and $\tilde{\lambda}$ are constants, $\dot{\omega}_x$ is the rolling acceleration (to be reduced) and I_{xx} is the longitudinal inertia of the ship. The increment in the moment μ_x seeks to counteract the hydrodynamic moments currently being generated by the motion of the ship through the sea, the so-called inertial moments, and the moments currently being generated by the fins. The constant of proportionality $\tilde{\lambda}$ used in equation (9.22) represents the learning parameters λ_y, λ_c and λ_σ. However, these parameters were defined for an error *e* given in terms of the difference between the desired and actual deflections, whereas the new error, $\Delta\mu_x$, is given in terms of the rolling moment. A new set of learning parameters $\hat{\lambda}_y$, $\hat{\lambda}_c$ and $\hat{\lambda}_\sigma$ is defined to account for this. This is not an added complication since they are determined heuristically in any case. They are small numbers, so that a gradual adjustment of the rules is generated at every time step. In this way, y_r, c_{ir}, and σ_{ir} are continuously changing while the ship is in motion.

To avoid local minima we can also add numerical momentum to the algorithm. To this end, at every time step, the following corrections are added to the fuzzy-logic parameters:

$$y_r(k+1) = y_r(k) + \kappa_y\left[y_r(k) - y_r(k-1)\right] \qquad \text{for} \quad r = 1\cdots R \tag{9.23}$$

$$c_{ir}(k+1) = c_{ir}(k) + \kappa_c\left[c_{ir}(k) - c_{ir}(k-1)\right] \qquad \begin{matrix}\text{for} \quad i = 1\cdots n \\ \text{for} \quad r = 1\cdots R\end{matrix} \tag{9.24}$$

$$\sigma_{ir}(k+1) = \sigma_{ir}(k) + \kappa_{\sigma}[\sigma_{ir}(k) - \sigma_{ir}(k-1)] \qquad \begin{matrix} \text{for} \quad i = 1 \cdots n \\ \text{for} \quad r = 1 \cdots R \end{matrix} \tag{9.25}$$

where κ_y, κ_c and κ_σ are the numerical moment coefficients for y_r, c_{ir}, and σ_{ir}, respectively. They are less-than-one constants, which are tuned experimentally.

The training procedure is done over a certain time window. The same seastate with the same initial conditions is run several times for the same time window. Each one of these runs is called an *epoch* or a cycle.

If the size of the time window is sufficiently large, the training process could be completed in just one epoch. This would be analogous to a real-life situation where the training procedure described here could be implemented in an actual ship. When using mathematical models (such as *LAMP* and *FINS*) computational limitations will determine whether to use one single large time window or several smaller windows in successive cycles.

Different sets of fuzzy-logic parameters could then be stored for different seastates, speeds and headings to be used as initial guesses to enhance the convergence when conditions change and the training procedure begins to search for a new optimal set. The same convergence procedure discussed in Chapter 5, section 5.5, can also be applied to the fuzzy-logic controller.

9.3.3 Initial-Rule Initialization. Clustering Algorithm

One of the advantages of fuzzy logic over other approaches (like neural networks) is that if some information about the target system is available, it can be used to initialize the fuzzy-logic scheme, provided this information can be expressed in the form of fuzzy-logic rules. Thus, for example, for a rudder-control system, the experience of a sailor who knows how to steer a certain ship could be translated into a set of rules from where the

backpropagation algorithm could start working in order to obtain a more refined set of rules. On the other hand, we always need an approximate set of rules to start the training of a fuzzy-logic scheme, and the information required to infer such rules is not always available. This may be the case of our fin-based, rolling-reducing control system. In this section, we describe how to generate this initial information to be used by the backpropagation algorithm as a starting set of rules.

First, we assume that we know the optimal antisymmetrical fin deflection $d(t) = \delta_d(t)$ that would provide the maximum rolling suppression. Later it is shown how to obtain initial training data other than δ_d.

To generate an initial set of rules, we utilize a particular implementation of the Clustering Algorithm [36]. The basis of this algorithm is to associate different sets of input data with certain outputs. In our case, this is equivalent to associating different sets of vectors **x** with different optimal fin deflections δ_d. To do this, an initial time window or epoch is run during which such data association is generated. For our problem, we actually perform the association process over the first two epochs. In the first epoch, the mean absolute value of each input datum $\bar{x}_i^0$ is recorded. In the subsequent epochs (or time windows) the following normalized input vector $\hat{\mathbf{x}}$ is used instead of **x**:

$$\hat{x}_i = \frac{x_i}{\bar{x}_i^0} \tag{9.26}$$

The same is done for the current and desired output data:

$$\hat{y} = \frac{y}{\bar{y}^0} \qquad \text{and} \qquad \hat{d} = \frac{d}{\bar{d}^0} \tag{9.27}$$

Next, a new vector $\hat{\mathbf{x}}' \in \Re^{n+1}$ is defined by adding $\hat{d}$ as an extra element of vector $\hat{\mathbf{x}}$ as follows:

$$\hat{\mathbf{x}}' = \left[\hat{x}_1,\ \hat{x}_2, \cdots,\ \hat{x}_n,\ \hat{d}\right]^{\mathrm{T}} \tag{9.28}$$

This expresses an association between $\hat{\mathbf{x}}$ and $\hat{d}$ such that $\hat{\mathbf{x}} \Rightarrow \hat{d}$. Let $\gamma_1 = \langle \hat{\mathbf{x}}'(1),\ \hat{\mathbf{x}}'(2), \cdots,\ \hat{\mathbf{x}}'(k), \cdots,\ \hat{\mathbf{x}}'(K) \rangle$ be a set of K vectors $\hat{\mathbf{x}}'$ in the $\Re^{n+1}$ space. Let's define another set of vectors $\gamma_2 = \langle \mathbf{c}'_1,\ \mathbf{c}'_2, \cdots,\ \mathbf{c}'_r, \cdots,\ \mathbf{c}'_R \rangle$ in the same space $\Re^{n+1}$. The sets γ_1 and γ_2 may or may not have elements in common ($\gamma_1 \cap \gamma_2 = 0 \ \vee \ \gamma_1 \cap \gamma_2 \neq 0$). We can compare sets γ_1 and γ_2 to generate other sets of vectors. One of such sets, β_r, can be obtained according to the following law:

$$\beta_r \subset \Re^{n+1}: \quad \text{If } \left|\hat{\mathbf{x}}'(k) - \mathbf{c}'_r\right| < \varepsilon \quad \Rightarrow \quad \hat{\mathbf{x}}'(k) \in \beta_r \qquad \begin{matrix} \text{for} & r = 1 \cdots R \\ \text{for} & k = 1 \cdots K \end{matrix} \tag{9.29}$$

where ε is a positive real number. Each set β_r defines a cluster of vectors $\hat{\mathbf{x}}'(k)$ that satisfies the inequality (9.29). Each cluster may have a different number of elements (vectors) N_r. The goal of the Clustering Algorithm is to generate sets of vectors β_r by comparing a set of vectors γ_1 with a set of vectors γ_2. To apply this algorithm, we start by choosing a first cluster β_1 during the first time step of the second epoch as:

$$\beta_1: \quad \mathbf{c}_1 = \hat{\mathbf{x}}(1) \Rightarrow \hat{d}(1) \tag{9.30}$$

The current number of clusters R is then set to one ($R = 1$). The same is done to the number of elements N_r belonging to cluster one ($N_1 = 1$). Next, for any subsequent time step k, the following algorithm is followed:

For $r = 1 \ldots R$:

$$\text{If } \left|\hat{\mathbf{x}}'(k) - \mathbf{c}'_r\right| \le \varepsilon \quad \Rightarrow \quad \hat{\mathbf{x}}'(k) \in \beta_r \tag{9.31}$$

$$N_r = N_r + 1$$

$$\text{If } \left|\hat{\mathbf{x}}'(k) - \mathbf{c}'_r\right| > \varepsilon \quad \Rightarrow \quad \text{Define a new Cluster} \tag{9.32}$$

$$R = R + 1$$

$$\mathbf{c}'_R = \hat{\mathbf{x}}'(k)$$

$$\hat{\mathbf{x}}'(k) \in \beta_R$$

The center of each cluster, $\mathbf{c}'_R$, constitutes the elements of the comparison set γ_2. Therefore, whenever a new cluster is defined, a new element is added to the comparison set. When the second epoch is over, we have R clusters of N_r elements each, where $K = \sum_{r=1}^{R} N_r$. The radius of influence of each cluster ε can be set by trial and error until the desired number of rules is obtained.

When all the time steps of epoch two (or second time window) are completed ($k = K$), the centers of each cluster are redefined as follows:

$$\mathbf{c}'_r = \frac{1}{N_r} \sum_{\hat{\mathbf{x}}'(k) \in \beta_r} \hat{\mathbf{x}}'(k) \qquad \text{for} \qquad r = 1 \cdots R \tag{9.33}$$

The clusters β_r constitute a set of initial rules. Each one of them can be utilized to define the membership functions of the fuzzy-logic rules. For our work, the first n elements of vector $\mathbf{c}'_r$, $\mathbf{c}_r = [c_{1r}, c_{2r}, \cdots, c_{nr}]^T$, define the initial centers of the Gaussian functions, whereas the last element of $\mathbf{c}'_r$, $\hat{d}_r$, characterizes the initial position of the singletons

associated with each rule. The initial spreads of the input Gaussian functions can be computed as:

$$\boldsymbol{\sigma}_r = \frac{1}{N_r} \sum_{\hat{\mathbf{x}}(k) \in \beta_r} |\hat{\mathbf{x}}(k) - \mathbf{c}_r| \tag{9.34}$$

where $\boldsymbol{\sigma}_r = \sigma_i$, for $i = 1 \cdots n$. To avoid anomalous big or small values for the spreads, the following limitations can be applied to the values of σ_i :

$$\text{If } \sigma_r < \frac{\bar{\varepsilon}}{\alpha} \quad \Rightarrow \quad \sigma_r = \frac{\bar{\varepsilon}}{\alpha} \tag{9.35}$$

$$\text{If } \sigma_r > \alpha \bar{\varepsilon} \quad \Rightarrow \quad \sigma_r = \alpha \bar{\varepsilon} \tag{9.36}$$

where $\bar{\varepsilon} = \frac{1}{R} \sum_{r=1}^{R} \left(\frac{1}{N_r} \sum_{\hat{\mathbf{x}}' \in \beta_r} |\hat{\mathbf{x}}' - \mathbf{c}'_r| \right)$ and $\alpha > 1$ is a constant.

The Clustering Algorithm, as described above, requires the optimal, or near optimal values of the fin deflections $\hat{d}(k) = \hat{\delta}_d(k)$ to operate. However, these data are not likely to be available. Nevertheless, it was found that the moment-matching procedure can be used to generate good initial estimates for the singleton coordinates of each rule. In accordance with this, the following data are passed to the Clustering Algorithm in order to generate the initial position of the singletons:

$$\hat{d}(k) = \frac{\dot{\omega}_x(k)}{\bar{\dot{\omega}}_x} \tag{9.37}$$

where $\bar{\dot{\omega}}_x$ is the mean absolute value of the rolling acceleration computed during the first epoch or time window. Even if the initial $\hat{d}(k)$ thus computed may be far from the final

values of the desired control law, their initial relative spread was similar to the final relative spread of the control-law output, for all cases explored. This allowed the backpropagation training algorithm to adjust gradually and effectively the locations of the singletons, $\hat{y}_r$, to their final positions, for all cases.

9.4 Results

The hull form and the inertia components used in the present examples are typical of a CG47 cruiser of the US Navy. It is the same ship used in Chapter 5. Let's summarize its main characteristics again. A pair of fins is attached to the hull amidships near the bilge area. The fins are tapered and have an aspect ratio of 2.2. The ratio of the tip-to-root chords is 0.8. Only the leading edge is tapered. The length of the root chord of the fins is 2% of the length of the ship. The fins have a dihedral angle of 15 degrees towards the bottom of the ship. The ship is moving at 20 knots and the seas are random. Also as in Chapter 5, the input data include the position of the surface of the sea at the four points distributed on the bow, stern and both sides of the ship (Figure 5-7).

In Figure 9-5 the Euler angle ϕ, called the "roll angle" in the figure, is plotted as a function of time for two cases. The light line represents the motion when there are no fins. The dark line represents the motion when there are fins, which are controlled by a fuzzy-logic controller with 58 rules. The controller has been trained over a window from 200 to 3000 time steps, during eight epochs. In this example, the ship is moving in a beam sea. As we did in Chapter 5, we define a ϕ-performance, ϕ_p, as

$$\phi_p = \frac{\sum_{k=1}^{K} |\phi(k)|_e}{\sum_{k=1}^{K} |\phi(k)|_0} \tag{9.38}$$

where k is the time step, K is the total number of time steps of each epoch, the subscript e denotes the current epoch and the subscript 0 denotes the initial epoch with no fins. Similarly, we can define a ω_x-performance. For the two plots shown in Figure 9-5, both the ϕ-performance ω_x-performance are 36%. Increasing the number of rules has a small impact in improving the performance of the controller as shown in Figure 9-6, were 2790 rules were used. For the two plots depicted in this figure, the ϕ- and ω_x-performance indices are 31% and 29%, respectively. We also trained the fuzzy-logic controller by using only nine rules. Even if such a controller is not as proficient as the controller trained with larger number of rules, it still exhibits a good effectiveness. We show this in Figure 9-7 parts *a* and *b*, where the corresponding ϕ- and ω_x-performance indices are of 42% and 43%, respectively.

In Figure 9-8 we show the controlled and uncontrolled results for the same ship in a random sea with a strong component of 45 degrees from the bow. The fuzzy-logic controller uses 87 rules. The ϕ-performance is 46%, and the ω_x-performance 29%. We detect a control effectiveness that is similar to what we observed for the beam-sea case. Increasing the number of rules does not produce a better performance. This is shown in Figure 9-9 where, for a 2790-rule controller, the ϕ-performance is 59%, and the ω_x-performance 37%.

For the results shown in this section we allowed the ship to move in all degrees of freedom, except in the x-direction. As we said in Chapter 5, part of our future work will be the study of the fins-propeller interaction, which would lead to a fully integrated propulsion-fin control system.

9.5 Concluding Remarks

In this chapter, we have designed an efficient fuzzy-logic controller to govern the motion of anti-rolling fins in ships. We have successfully implemented the same moment-matching procedure we used to train the (rolling) neural-network controller in Chapter 5. This training approach could be applied on line on a real ship. The roll reduction we obtained with the fuzzy-logic controller was comparable to what we obtained in Chapter 5 for the neural-network controller. We have explored the use of different numbers of rules for the fuzzy-logic controller. We obtained very good roll reduction by using about 60 rules, though we still obtained good results with only nine rules. Larger numbers of rules did not improve the efficiency of the controller. The number of epochs to train the fuzzy-logic controller and the neural-network controller were comparable (between 10 to 20, for a 3,000 time-step window and random seas). We detected a larger initial response reduction of the fuzzy-logic control system compared to the response reduction obtained with neural-network control system (about 20% in favor of the fuzzy-logic system). This is due to the fact that the fuzzy-logic controller is provided with a set of effective initial rules, which are generated by the clustering algorithm, whereas the initial performance of the neural-network controller is determined by a random initialization of the neural-network weights.

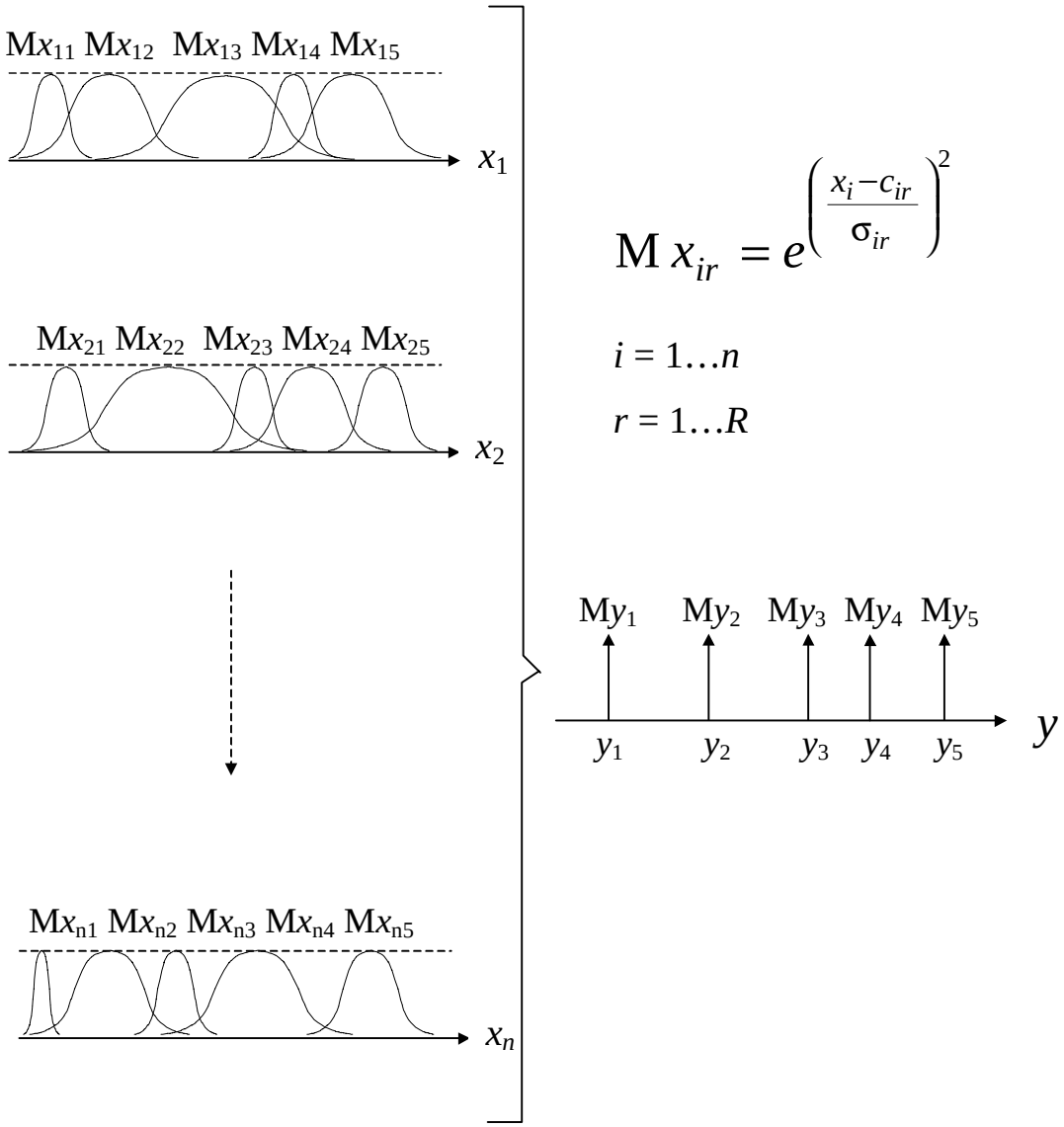


Figure 9-1: Input/output membership functions. Five rules (R = 5); n input data.

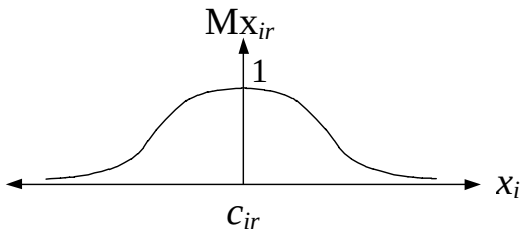


Figure 9-2 Gaussian membership functions.

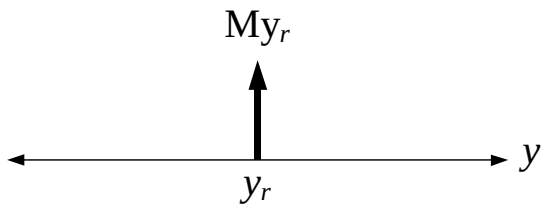


Figure 9-3 Singleton membership functions.

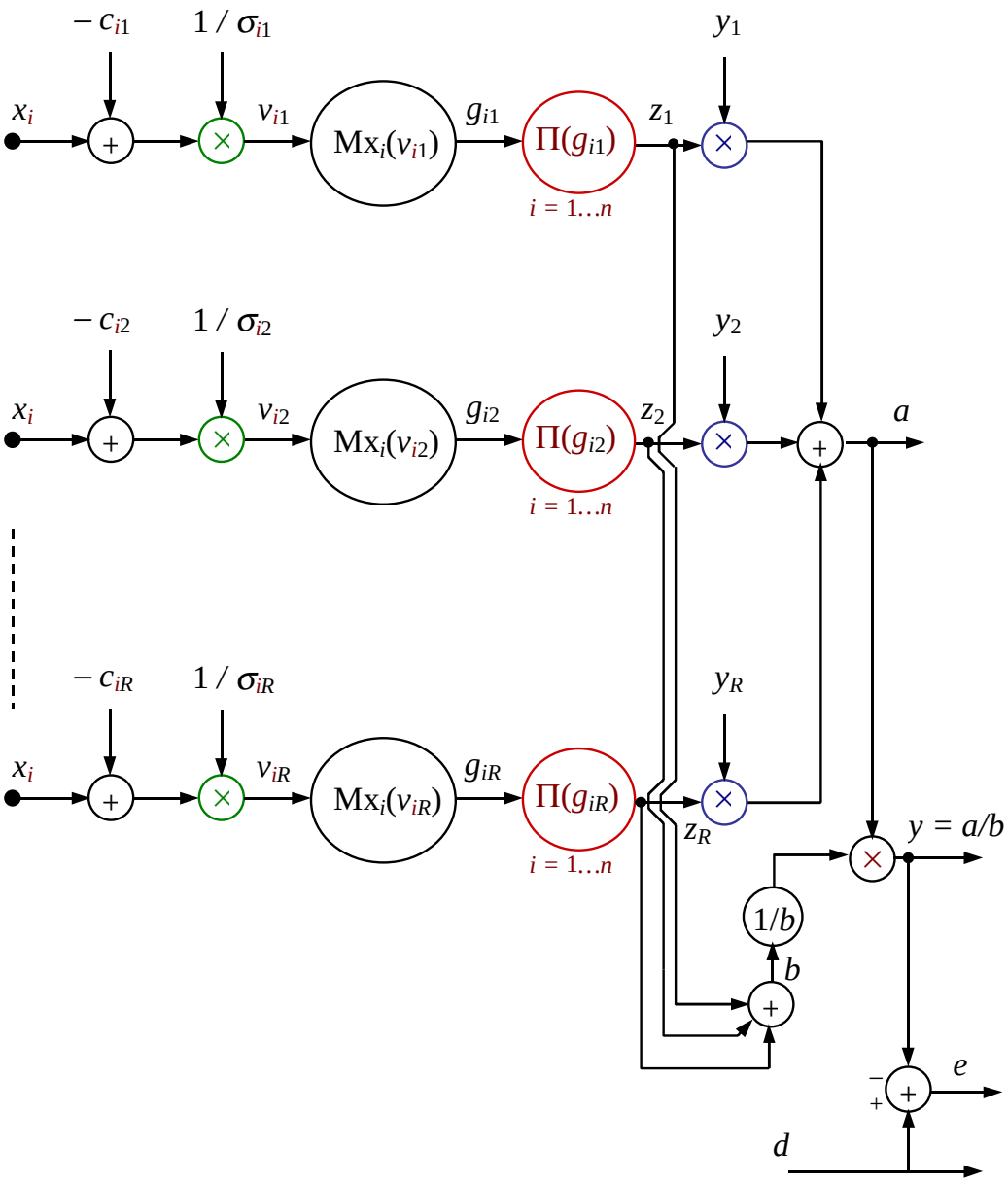
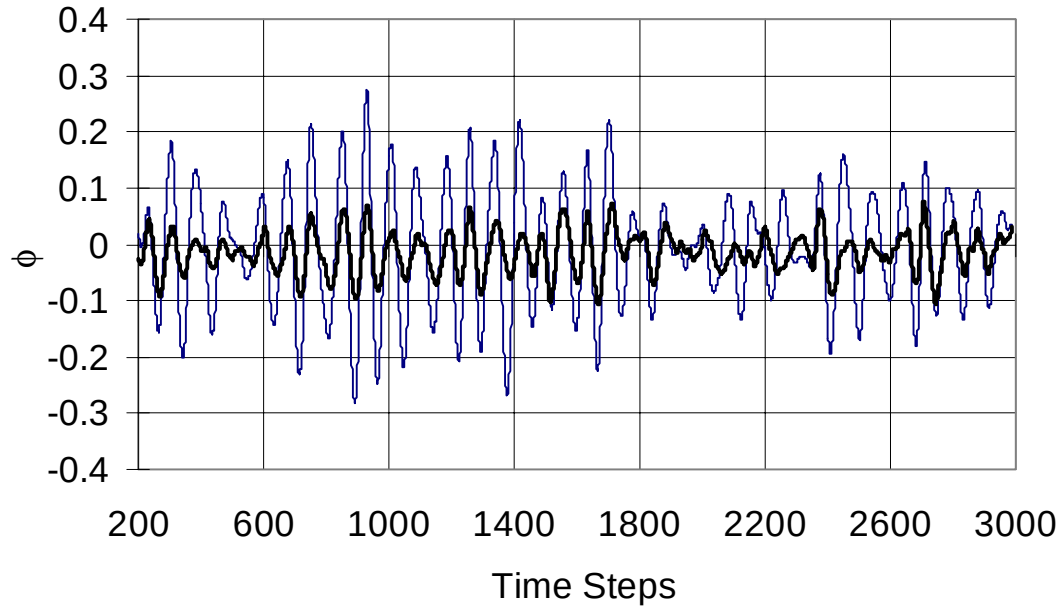
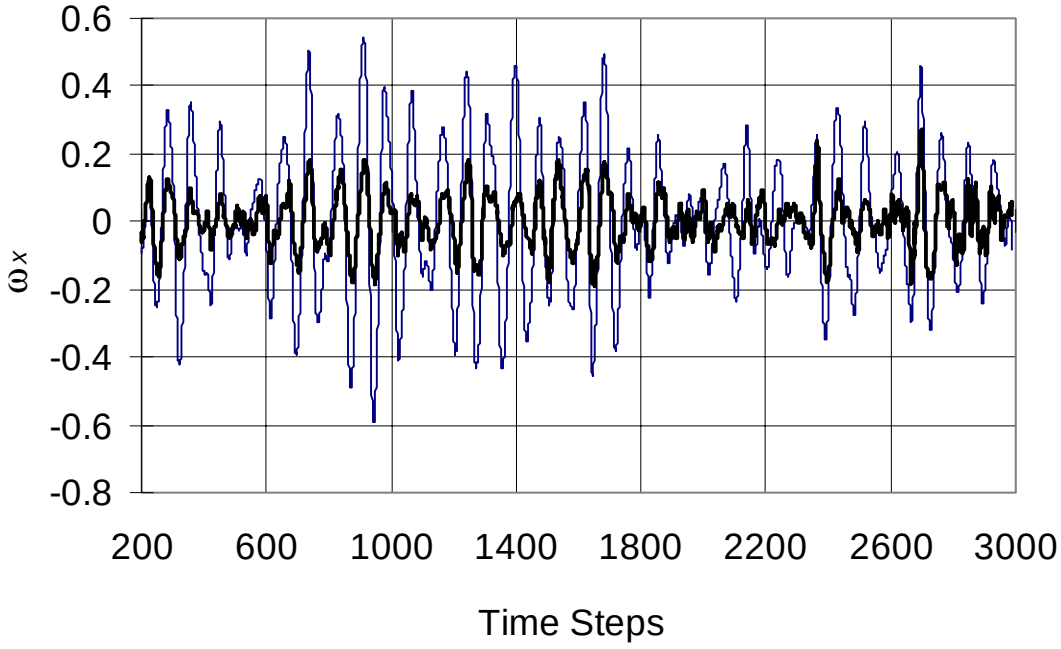


Figure 9-4: Fuzzy logic system.

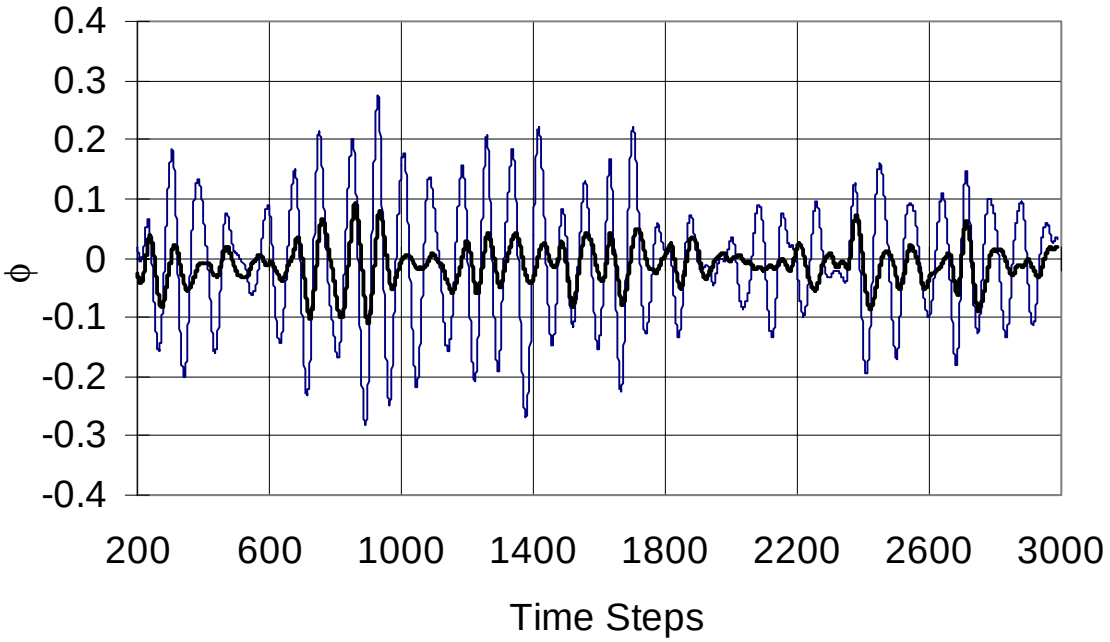


(a)

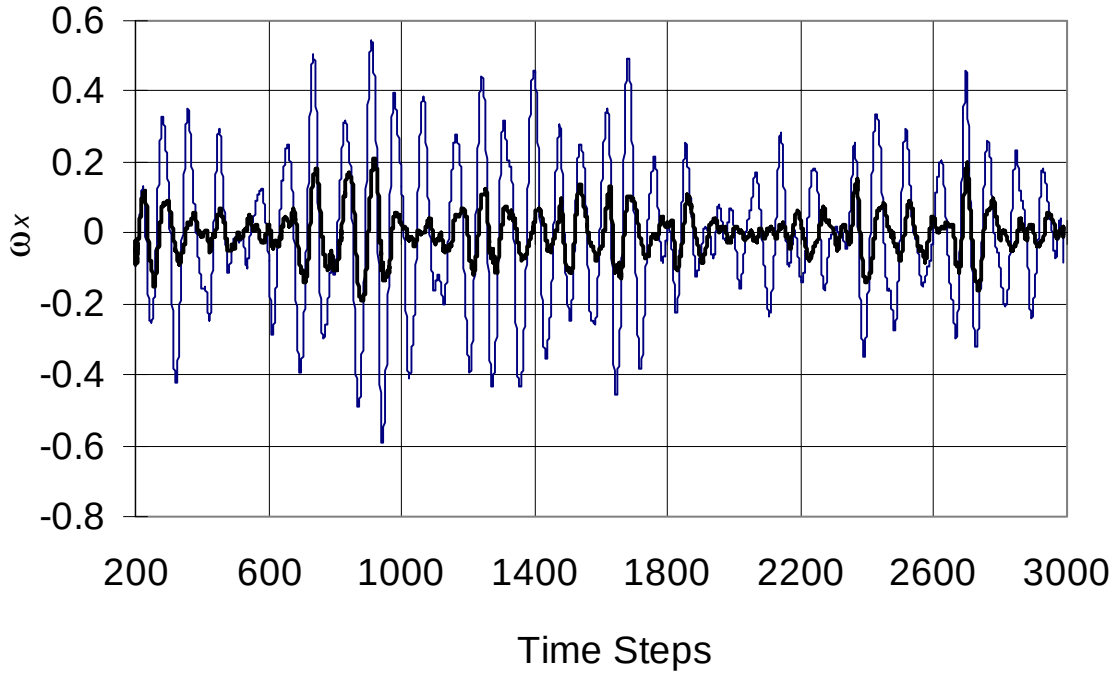


(b)

Figure 9-5: (a) Roll angle as a function of time; (b) roll rate as a function of time. Thin curves: no fins. Thick curves: active fins, 58 rules. Random beam sea.

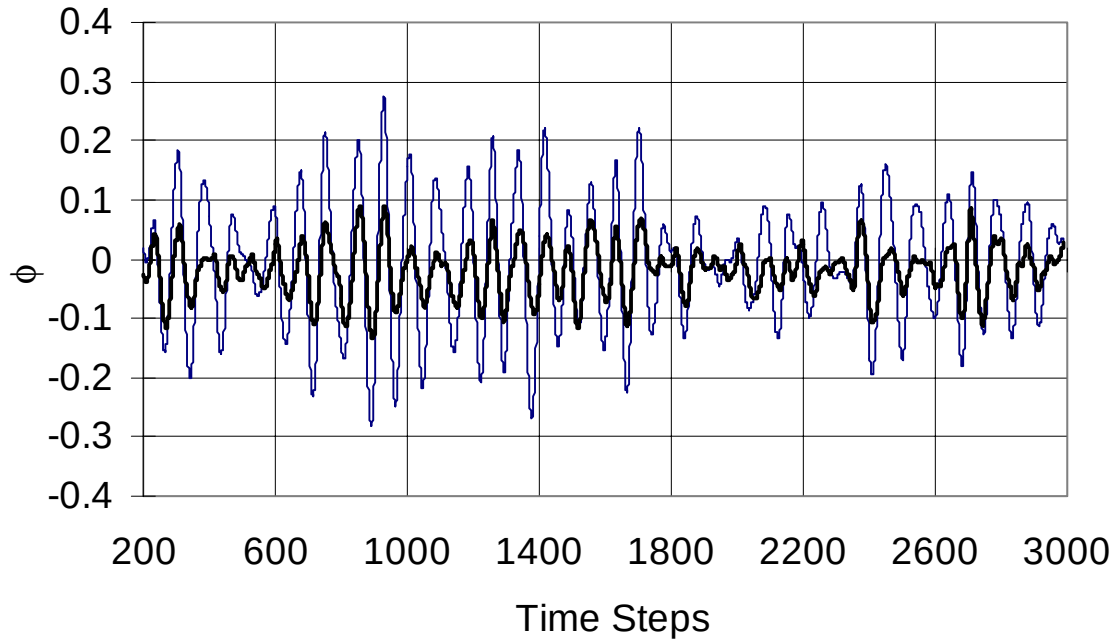


(a)

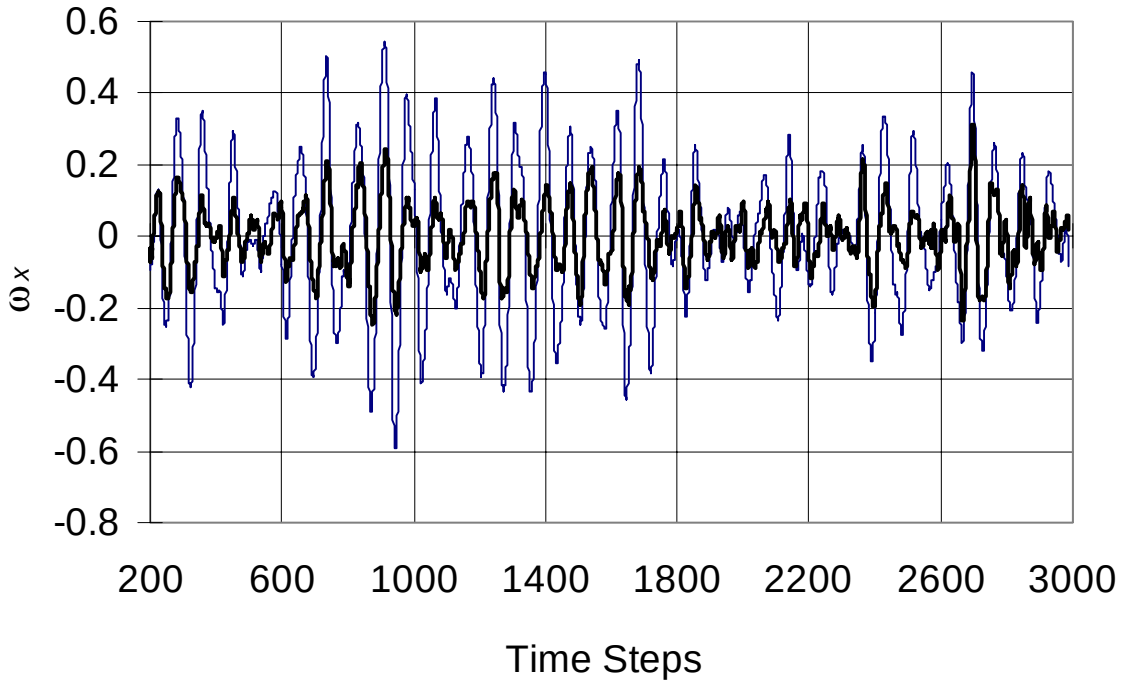


(b)

Figure 9-6: (a) Roll angle as a function of time; (b) roll rate as a function of time. Thin curves: no fins. Thick curves: active fins, 2790 rules. Random beam sea.

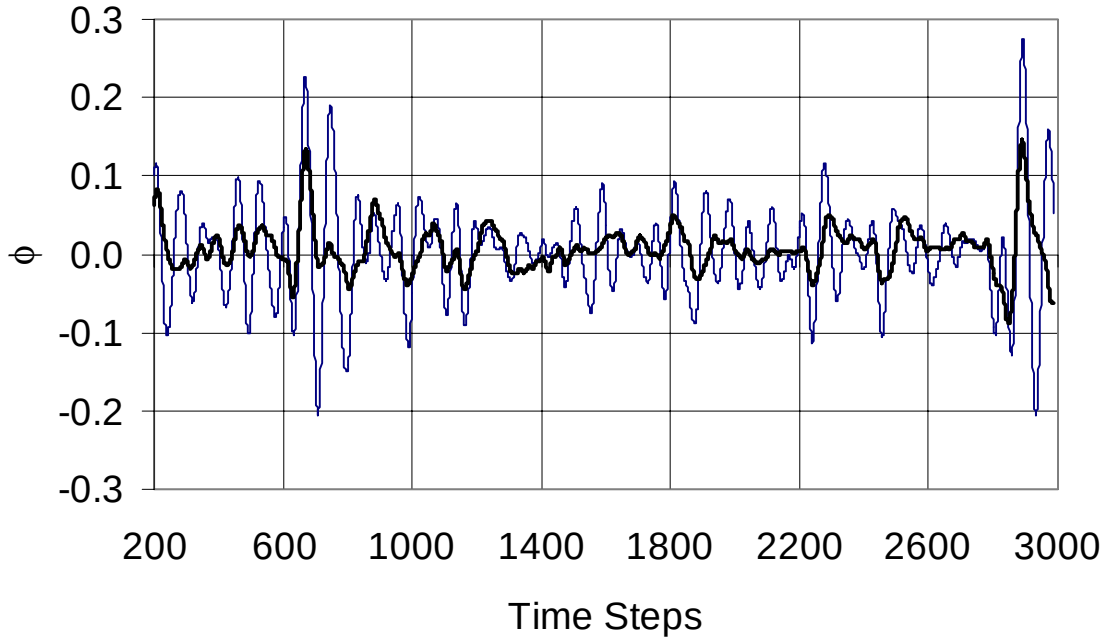


(a)

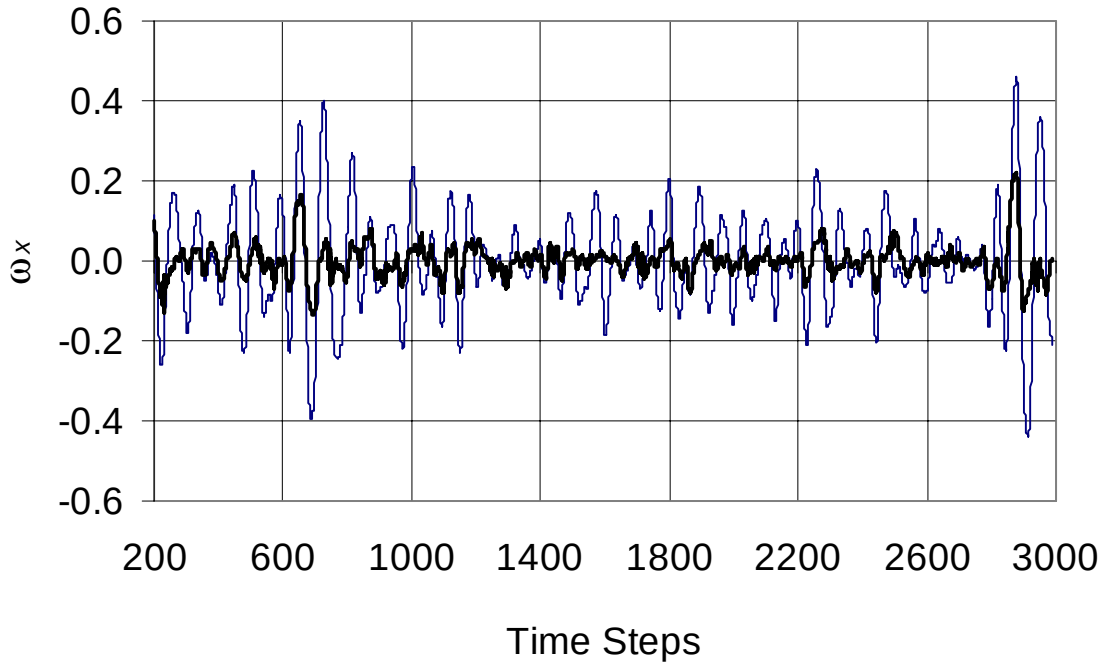


(b)

Figure 9-7: (a) Roll angle as a function of time; (b) roll rate as a function of time. Thin curves: no fins. Thick curves: active fins, 9 rules. Random beam sea.

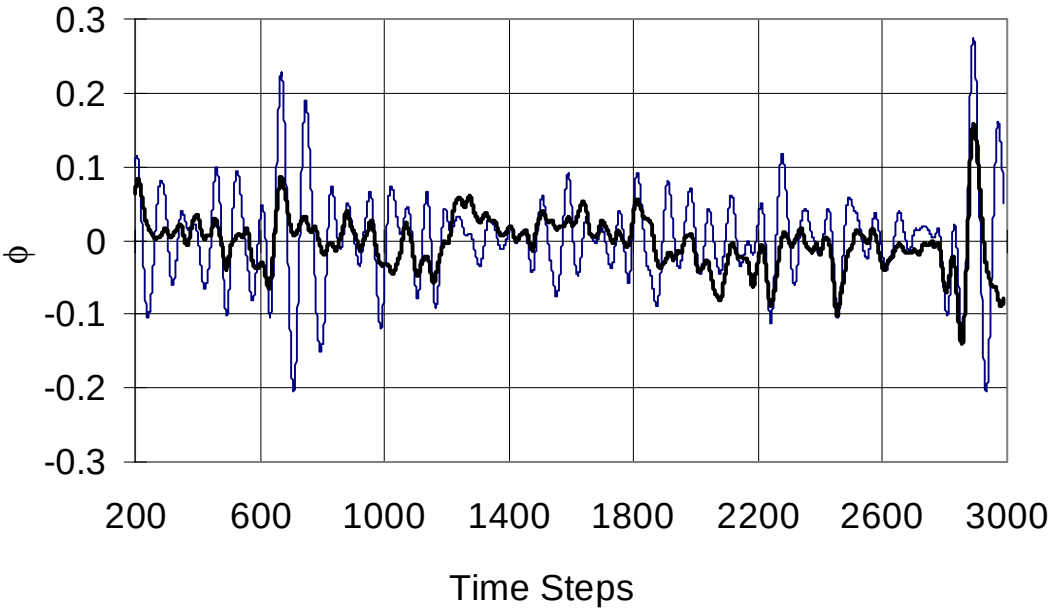


(a)

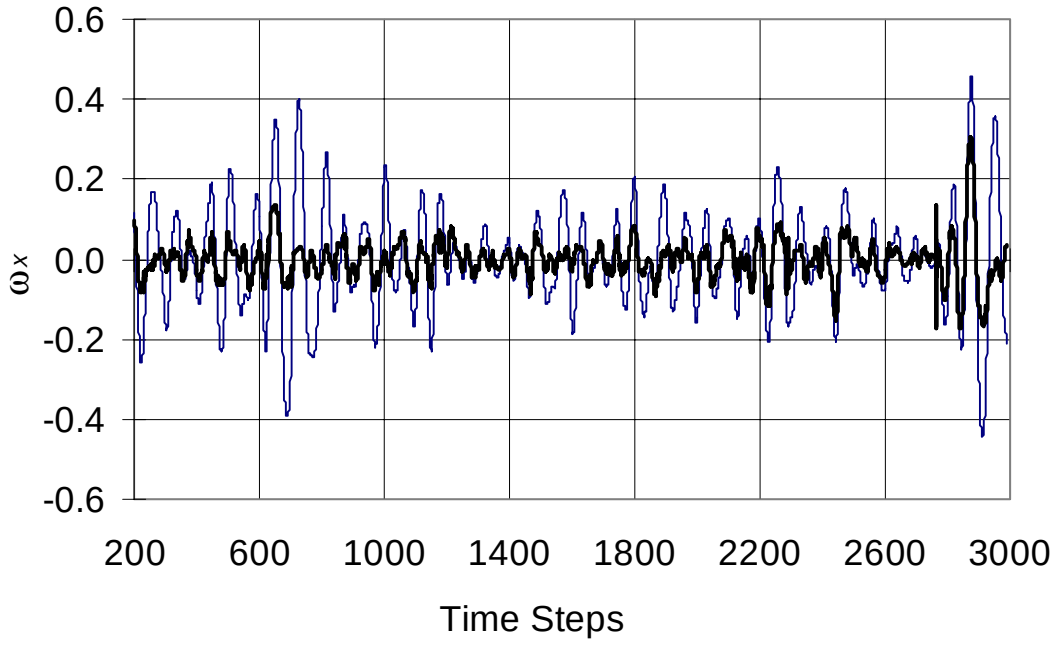


(b)

Figure 9-8: (a) Roll angle as a function of time; (b) roll rate as a function of time. Thin curves: no fins. Thick curves: active fins, 87 rules; 135 degrees random sea.



(a)



(b)

Figure 9-9: (a) Roll angle as a function of time; (b) roll rate as a function of time. Thin curves: no fins. Thick curves: active fins, 2790 rules; 135 degrees random sea.