Genetic Adaptive and Supervisory Control *

La Moyne L. Porter II and Kevin M. Passino †
Department of Electrical Engineering
The Ohio State University
2015 Neil Avenue
Columbus, Ohio 43210

Abstract

A genetic algorithm (GA) uses the principles of evolution, natural selection, and genetics to offer a method for parallel search of complex spaces. Motivated by past work using the GA as a tool for off-line computer-aided-design of control systems, we introduce several ways to use the GA for on-line identification and controller tuning. In particular, we introduce two new, but closely related approaches to genetic adaptive control which we call "genetic model based control" (GMBC) and "genetic model reference adaptive control" (GMRAC). In these techniques a GA manipulates a set (population) of parameterized controllers in order to evolve the controller most capable of providing good performance for the current plant operating conditions. Next, we introduce a hierarchical GA-based supervisory controller (GASC) that seeks to evolve the GMRAC's fitness evaluation procedure so that the controllers chosen by GMRAC guide the system towards reduced error and decreased use of control energy. The use of GASC removes the need for the designer to specify GMRAC parameters. In addition, we show how GA-based estimation (GAE) can be used to evolve a model of the plant that is used in the model based controllers GMBC and GMRAC. A cargo ship steering problem is used throughout this paper as a theme example to illustrate each of the genetic adaptive control concepts and techniques.

Keywords: Adaptive control, genetic algorithms, supervisory control.

---

*This work was supported in part by National Science Foundation Grant IRI 9210332. **Bibliographic information for this paper: Porter L., Passino K., "Genetic Adaptive and Supervisory Control," Int. Journal of Intelligent Control and Systems, Vol. 2, No. 1, pp. 1–41, 1998.**

†Please address all correspondence to K. Passino; (614) 292-5716, email: passino@ee.eng.ohio-state.edu.

1 Introduction

Since the inception of the genetic algorithm (GA) concept by Holland [1] in 1975 it has been useful in solving a wide variety of problems. Economics, game theory, and the traveling salesman problem are just a few instances of situations where the GA has been used to prize an optimal solution from a complex, nonlinear search space [2, 3]. The GA has also found application in the area of design automation for conventional and intelligent controllers. Typically, for this, a controller is decomposed into a set of parameters which the GA attempts to optimize by using simulation based fitness evaluation of candidate controllers in the closed-loop system. For instance, Lee and Takagi [4] design a fuzzy system using a genetic algorithm for the inverted pendulum. Their controller fitness evaluation is based upon simulation of the system over a variety of initial conditions to obtain a fuzzy controller capable of handling a variety of operating conditions. Their GA manipulates strings which represent input and output membership functions. Their fitness evaluation incorporates a strategy to minimize the number of rules and "scores" the ability of the fuzzy controller to balance the pendulum. Porter and Borairi [5] use the GA in an eigenstructure assignment technique. Their GA chooses values of a linear feedback matrix to minimize the error between actual closed-loop eigenvalues and desired eigenvalues as well as actual and desired eigenvectors. Michalewicz, et al. [6] use the GA to solve certain optimal control problems. Ishibuchi, et al. [7] design fuzzy controllers for pattern classification with the GA attempting to minimize the number of rules while maximizing the number of correct classifications. Katai, et al. [8] present a technique which utilizes a GA and a fuzzy controller to reduce the error between a model of the system and the actual system over a time window. Karr and Gentry [9] use a GA to design a fuzzy controller to control the pH of an acid-base system. Park, Kandel, and Langholz [10] optimize a fuzzy reasoning model via a genetic algorithm to control a direct current series motor. Varšek, et al. [11] use a GA to derive and subsequently optimize rules for the control of an inverted pendulum. Nomura, et al. [12] present a method for GA tuning of a fuzzy controller that fits input-output data. They utilize a gradient descent method coupled with rule minimization to obtain optimal input membership functions. Das and Goldberg [13], Maclay and Dorey [14], Kristinsson and Dumont [15], and Etter, et al. [16] use the GA for system identification. Yao and Sethares [17] use the GA for nonlinear parameter estimation.

Off-line design schemes often require many simulations to validate an acceptable controller and results are generally obtained at the expense of a large amount of computation. An exhaustive study of a plant (especially a very nonlinear one) with respect to disturbances and reference inputs is usually not feasible and off-line design by a GA may produce a control system which is only effective for the operating conditions studied. Despite these disadvantages, the GA has a great deal of flexibility in that any ideas of how to control, identify, or estimate a system which can be codified by a string representation are amenable

to search by the GA (and any system that can be parameterized can have its parameters represented by strings). The references above are a testament to the GA's breadth of usage. In this paper we show how GA's can be used in *on-line* adaptive control and system identification to evolve the control system that performs well under the current operating conditions. To do this we exploit the flexibility of the GA found in off-line design within a framework which allows the GA to make decisions quickly (within a sampling instant) so that evolution occurs in real-time.

The development of the genetic adaptive control techniques and ideas is presented in a constructive manner where later techniques build on earlier ones. First, in the genetic model based control (GMBC) approach a GA uses (i) a model of the plant, (ii) current and past input-output data, and (iii) an estimate of future plant outputs to decide which controller in a population of controllers should be used to control the plant at each time step. Genetic model reference adaptive control (GMRAC) uses (i) - (iii) for GMBC along with a "reference model" [18, 19] that is used to characterize the desired performance of the closed-loop system. GMRAC fitness evaluation is based upon the ability of a controller to meet user-specified design objectives. GA-based supervisory control (GASC) acts as a supervisor for GMRAC and has a goal to decrease system error and use of control energy by manipulating GMRAC's fitness function. GASC provides a method to adaptively tune GMRAC so that manual design iterations can be avoided. GA-based estimation (GAE) evolves the model in (i) above when no mathematical model of the plant is available for GMRAC. The use of GAE coupled with MRAC bears some conceptual similarity to the work in [15]. There the authors use the GA as an on-line identifier for the parameters of a linear system then use the parameter estimates in a certainty equivalence control law [18, 19] to specify the controller parameters (i.e., "indirect adaptive control"). In our approach we use one GA to identify the plant parameters but use the parameter estimates in the fitness evaluation in another GA in GMRAC rather than using a certainty equivalence approach. Overall, then we study both "indirect" and "direct" methods for genetic adaptive control (in "direct" adaptive control we identify the controller parameters that will result in good closed-loop behavior without estimating the plant parameters). An early version of this paper appeared in [20] where only the GMRAC is discussed.

Note that we use a base-10 GA (i.e., one that operates on a string of digits between 0 and 9) in all of our adaptive schemes since (i) it provides a natural representation for the controller and model parameters to be manipulated and (ii) no encoding and decoding of parameters is necessary as it is for base-2 GA's [2]. We emphasize, however, that our techniques do not depend on the use of the base-10 GA; a standard base-2 GA will work in a similar manner. Moreover, a floating point based GA will work similarly. Also note that due to past investigations in conventional adaptive control [18, 19], we have a good idea of how to define the necessary controller and model structures that we will tune in the adaptive controller. Hence, the fixed length structures typically

associated with the GA can be employed and there is no need for the more general representation proposed in [21, 22]. Next, we provide a brief overview of this paper.

In Section 2, we give an overview of the base-10 genetic algorithm which is used throughout this paper. Section 3 describes the use of the GA in the GMBC and GMRAC adaptive schemes. In particular, we begin by developing the GMBC technique, illustrate its use in a cargo ship steering application, and provide broad guidelines on how to design genetic adaptive controllers. Next we introduce the GMRAC technique and show how the design guidelines we have established can be used to design a GMRAC for the cargo ship. We evaluate the performance of the technique by studying (i) the effects of a rudder disturbance, (ii) ship speed variations, (iii) computational delay in the adaptive loop, and (iv) the use of reduced order models in GMRAC. The performance of GMRAC is compared to conventional model reference adaptive control (MRAC) and fuzzy model reference learning control (FMRLC) by using the results in [23]. Section 4 develops GASC, GASC with "population splitting" and then describes the application of GASC to the cargo ship steering application. Section 5 illustrates the combined use of GASC, GMRAC, and GAE for the cargo ship steering application. Finally, Section 6 provides concluding remarks as well as suggestions for modification of GA implementation to possibly improve performance and recommendations for further GA adaptive control studies.

2 Background: A Base-10 Genetic Algorithm

The GA performs a parallel search of a parameter space by using *genetic operators* (e.g., *selection, crossover, and mutation*) to manipulate a set of encoded strings which represent system parameters[1]. These genetic operators combine the strings in different arrangements where the optimal configuration being sought is one which maximizes a user specified *objective function* (also called a "fitness function"). The parallel nature of this search is realized by the algorithm's repetitive processing of a *population* (set) of strings beginning with an initial population. This initial population is either a set of guesses of potential solutions to the optimization problem or a random set of strings generated by the computer. A subsequent population is created via evaluation of the objective function and the use of genetic operators to form a new *generation* of strings which hopefully comprise the best characteristics of the previous set. Ideally, the strings of the new generation are either as capable or more capable of maximizing the value of the objective function than those of the previous population. Typically, the strings that maximize the objective function at the time of termination of the GA are taken to be solutions to the optimization problem.

[1]While a brief overview is provided, we assume that the reader has a familiarity with the conventional base-2 GA for which there exist many excellent tutorial introductions (see e.g., [2, 3, 24]).

A string is composed of digits (genes) each of which can take on different values (alleles). In our artificial genetic environment we can use alphabets of any cardinality we desire in order to encode these values. In a binary environment, we can represent an allele with a 0 or 1. The reproduction operation merely copies selected strings from the old generation into the new generation. Strings are selected for reproduction based upon their fitness values; thus strings with higher than average fitnesses are preferentially copied into the new generation. Goldberg [2, p. 11] cites the analogy of spinning a roulette wheel partitioned according to the fitness of each individual string with respect to the average fitness of the entire population. Thus, strings with large fitness values occupy a greater portion of the wheel and are more likely to be selected. The crossover operation is the primary vehicle for developing new structures. Crossover qualifies as a genetic operator since it allows for the exchange of chromosome building blocks (genes) which occurs in natural genetics. Once two strings are selected by the reproduction operation, crossover will occur with a probability, p_c, which is specified by the user during initialization of the routine. If crossover occurs, a "cross site" is randomly determined. This cross site is a number between 1 and $p-1$, where p is the length of the string, which determines how much genetic material will be exchanged between the two selected strings. Once the cross site, k, is determined, crossover dictates that the two strings simply exchange the alleles between the $k+1$ position and the end of the string.

The mutation operator is the secondary method for introducing new structures into the population. The mutation operation is performed on a digit-by-digit basis: each digit (position) of the string has an equal probability of mutation, p_m, against which it is tested. When mutation occurs, the string position is changed to a different allele selected from the set of possible digits. Mutation should be used sparingly (by choosing p_m to be small) as increased use results in a random walk through the search space.

The operation of the GA changes slightly depending on the base of the numbers to which we apply the genetic operators. Traditionally GA's have been designed to operate over binary numbers (we refer to as "GA_2") and more recently there have been several base-10 GA's ("GA_{10}") developed [3]. To avoid the need for encoding and decoding of strings we will employ a GA_{10} algorithm that operates similarly to the GA_2 described in [2] except: (i) its digits vary over the numbers $0, 1, 2, \ldots, 9$ and there is an extra digit for the "+" or "-" sign, (ii) we split the strings into a portion to the left and to the right of the decimal point, and (iii) its genetic mutation operator randomly perturbs the digits to any value (including the value it currently has) in $0, 1, 2, \ldots, 9$ or toggles between "+" and "-" for the sign digit with probability, p_m. If strings outside the domain are generated by the mutation and crossover operators then we generate another candidate via these operations. For pathological cases, a limit is placed upon the number successive mutations allowed and in violation of this limit, we reset the parameter (and string) to its maximum or minimum value (whichever is closest to the actual value of the string). An alternative

approach would be to simply pick the value for the string to saturate along the boundary of the allowed range of the parameter. It is important to note that the approaches presented in this paper do not depend in any critical way on using GA_{10}. The standard GA_2 method works in a similar manner (we have done some simulation-based investigations to verify this for the maximization problem in [3, p. 18]). We use GA_{10} to simply avoid issues in coding and decoding strings and to make an algorithm for which it is easy to gain insight into its search procedure. For instance, we find it easier to gain intuition about the operation of the GMBC and GMRAC since the underlying GA operates over base-10 numbers rather than long binary strings.

3 Genetic Adaptive Schemes

3.1 Genetic Model Based Control

In genetic model based control (GMBC), which is shown in Figure 1, we use a GA that tunes an underlying controller by using a mathematical model of the plant. Receiving plant output and reference input information, the GA selects the "best" controller at each time step from a "bank" of candidate controllers (denoted by "C") by using an objective function to evaluate the suitability of each controller. It is important to emphasize that adaptation occurs between one time step and the next where one time step corresponds to one generation of GA operation[2]. Learning is only taking place in the sense that the population of strings is shaped by GA operators over successive generations. Controllers that provide "good" closed loop control are retained within the population while "bad" controllers are removed from the genetic pool thus improving the ability of GMBC to control the plant. Next, we provide a detailed explanation of each component of the GMBC in Figure 1.

3.1.1 Functional Architecture

The plant is defined by a nonlinear differential equation

$$\dot{x}(t) = f(x(t), u(t), d(t)) \quad (1)$$

$$y(t) = g(x(t), u(t), d(t)). \quad (2)$$

where $u(t)$ is the control input, $d(t)$ is the disturbance input, $x(t)$ is the state, and $y(t)$ is the plant output (all quantities can be vectors or scalars as necessary).

[2]Real-time implementation issues must be considered. It will become clear that we will fold one step of the GA's operation into one time step in discrete time. Since the computational resources for computing one generation in a GA can for many applications be quite minimal we gain an ability to implement genetic adaptive control in real time. We have, in fact, implemented GMRAC in our lab on a 486 PC for the control of a tank system. We will discuss issues of computational complexity in more detail later in this paper.

Let $y(kT) = y(t)$ where T is the sampling period, be the sampled output of the continuous time plant. Let $y(k), \ldots, y(k-N_y)$ denote the sampled plant outputs from time k to time $k - N_y$ where N_y is a fixed positive integer (notice that we often omit the sampling period T in our notation). Similarly, we use the fixed positive integers N_u and N_r to indicate the number of past values of u and r we will consider.

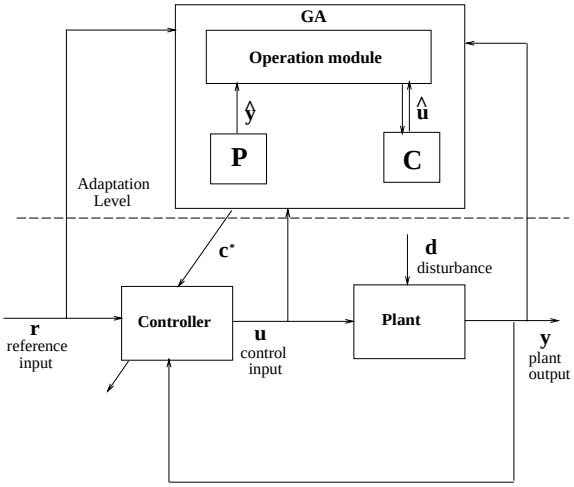

Figure 1: Genetic Model Based Control

We will use a model of the plant, P which we denote by

$$\hat{x}(k+1) = \hat{f}(\hat{x}(k), \hat{u}(k)) \quad (3)$$

$$\hat{y}(k) = \hat{g}(\hat{x}(k), \hat{u}(k)) \quad (4)$$

where $\hat{u}(k)$ is the control input, $\hat{x}(k)$ is the state, and $\hat{y}(k)$ is the output in the adaptation level of GMBC shown in Figure 1.

Multiple steps into the future associated with any of the variables $\hat{y}$, $\hat{r}$, or $\hat{u}$ will be demarcated by N_f as in $\hat{y}(k), \ldots, \hat{y}(k + N_f)$. GMBC uses P and candidate controller outputs $\hat{u}$ to determine estimated plant outputs $\hat{y}(k)$. GA processing occurs in the "operation module" shown in Figure 1. It uses r, u, y (and their past values), and the plant model P to evaluate the fitness of the strings in the population C of candidate controllers. At each time step (i.e., each generation) the GA chooses the $c^* \in C$ with maximum fitness value to control the plant from time k to time $k + 1$ (hence $c^* \in C$ always denotes the controller that is currently being used).

3.1.2 GA Operation Module

In addition to the three GA operators *selection, crossover*, and *mutation*, we will also utilize an operator called *elitism* [2, p. 115]. Elitism ensures that the string with the largest fitness value will propagate to the next generation without manipulation by other GA operators. Elitism is used since it is likely that within a sufficiently small time range (i.e., a small number of generations), one candidate controller (i.e., $c^* \in C$) will be better than other controllers. To perturb the parameters of the best controller unnecessarily may result in an unsatisfactory performance for which no genetic technique can adapt. Hence, using the elitism operator, over a certain time range the best controller will consistently determine the control signal to the plant. The string which defines the best controller is given a number of copies in proportion to its fitness relative to the average fitness of the population. The remaining slots, if any, are chosen as in the previous sections (i.e., via *selection, crossover*, and *mutation*).

The use of elitism fits well with genetic adaptive control when the nature of its operation is considered. We are not primarily concerned with finding a global optimum for this system (i.e., a set of controller parameters which will perform perfectly over a wide range of plant states, reference inputs, and time ranges) but more of an instantaneous local optimum which will satisfy performance criteria between time k and time $k+1$. It is interesting to note, however, that we do in fact obtain a control system which performs well over a wide range of plant states, reference inputs and time ranges even though we are optimizing in a local fashion. Because the time range of optimization is short (i.e., per generation) and we do not have the stringent global optimum criteria as for a typical problem in off-line design [4] or a maximization problem [3], we are free to have a relatively large crossover and a particularly large mutation probability. The large crossover and mutation rates will greatly aid in the adaptive nature of this scheme; however with these large rates, the probability of every string being changed in every generation is large. While in general no convergence of the genetic algorithm is guaranteed, global and even local optimization is hindered by excessive operator manipulation. We cannot afford to gamble that the very next generation will produce a controller of a comparable or greater aptitude to control the plant; with elitism we are ensured that our best idea of how to control the plant will remain in the genetic pool.

It is interesting to note that we have found that larger than normal mutation probabilities are sometimes useful in on-line use of GAs (a similar characteristic was found in [17] where the authors used '"extinction and migration" to essentially achieve a very high mutation rate). The higher mutation rates are sometimes necessary since unlike many off-line optimization problems the on-line approach will use a fitness function that changes over time. Intuitively, then it is necessary to more broadly search the space when the fitness function you are trying to optimize is continually changing (i.e., the GA is continually seeking to maximize a fitness function that is continually changing – to do this

is has to be more liberal in its search strategy).

Let C_i denote the ith controller (a string of digits parameterizing the controller structure) in the population of controllers C shown in Figure 1 and assume that the size of the population $|C|$ is n. The objective function utilized in GMBC is of the form, $J_i = \sum_{j=1}^{N_p} \alpha_j {p_j}^2$ where p_j is a parameter used to evaluate the "goodness" of C_i, N_p is the number of those parameters, and the α_j are scaling factors. For example, p_j might represent the amount of error between the reference signal and the plant output. We wish J_i to approach zero as the algorithm operates so to form the fitness function that the GA maximizes we use:

$$\bar{J}_i = \begin{cases} \frac{1}{|J_i|} & J_i \neq 0 \\ 2 * \max(\bar{J}_i) & J_i = 0 \end{cases} \tag{5}$$

where $i = 1, 2, \ldots, n$ and n is the population size. In the rather unlikely event that a J_i has a fitness of zero, the second equation in 5 identifies this J_i as the "elite" string by mapping it to the largest fitness value. We choose the quantity $2 * \max$ here more or less arbitrarily. Assigning a fitness value that is too large would result in the next generation completed dominated by this "zero" fitness valued string. This may not be desirable in all cases, so the $2 * \max$ represents an acceptable trade-off. Once a $\bar{J}_i$ has been assigned to all strings, selection proceeds normally with $\bar{J}_i$ used as the fitness value.

Suppose that the candidate controllers are of the proportional-derivative (PD) form and we allow the GA to pick the proportional (K_p) and derivative (K_d) gains. In this case, GMBC proceeds according to the following pseudo-code (the pseudo-code changes only slightly for general linear or nonlinear controllers):

1. Initialize the GA. Choose the number of digits to represent each controller parameter K_p and K_d. Choose crossover probability p_c and mutation probability p_m. Generate an initial population of K_p and K_d gains (we make a random selection; however, if one has good a priori knowledge of how to choose K_p and K_d then the initial population can be seeded with a better set of candidate controllers). Initialize sample time, T. Set time, t, to zero. Set initial conditions for the plant and the model of the plant P. We choose all initial conditions to be zero.

2. Collect $r(k)$ and $y(k)$.

3. Generate $\hat{u}(k)$, for each population member C_i, $i = 1, 2, \ldots, n$ using the PD control law $\hat{u}(k) = K_p e(k) - K_d e_{dt}(k)$ where $e(k) = r(k) - y(k)$ and $e_{dt}(k) = \frac{e(k) - e(k-1)}{T}$. Then generate $\hat{y}(k+1)$ using equations 3 and 4 ($\hat{x}(0) = 0$). Thus the architecture of the "operation module" can be modeled as in Figure 2. Note that $r(k)$ represents our closest approximation to $\hat{r}(k+1)$ and in this paper we always choose $\hat{r}(k+1) = r(k)$.

4. Assign fitness to each element of the population C_i, $i = 1, 2, \ldots, n$:

   Let $p_1 = e(k) - e(k-1)$ (we want all controllers to minimize this error).
   Let $p_2 = \hat{r}(k+1) - \hat{y}(k+1)$ (estimated amount of error caused by the candidate controller).
   Let $p_3 = \hat{u}(k)$ (estimated control energy employed by candidate controller).

   $J_i = -(\alpha_1 p_1^2 + \alpha_2 p_2^2 + \alpha_3 p_3^2)$ Notice that this fitness function can vary significantly over time since the environment of the plant and the plant itself can change over time. The fitness function must capture such dynamical changes so that it can evolve a new set of controllers for the new conditions.

5. The maximally fit C_i becomes c^*. This c^* is the controller used between times k and $k+1$.

6. Produce the next generation using GA operators.

7. Let $t := t + T$. Go to Step 2.

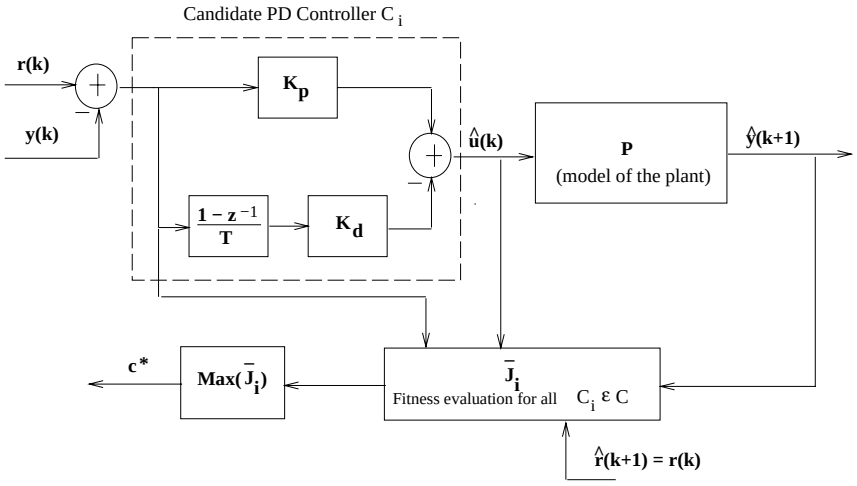

Figure 2: GMBC Operation Module

By studying the GMBC pseudo-code it should be apparent that the amount of computation required to compute a control action is not excessive. Steps 2 - 7 execute during the operation of the GMBC. Step 2 is executed for any control algorithm and is simple. In Steps 3 - 5 we compute the value of $\bar{J}_i$ for the n candidate controllers (hence the population size n must be chosen based upon

the sampling period T). This involves taking one time step forward using the model P and the candidate controller C_i (as the designer chooses the structure of the controller it is typically of a reasonable order) and is not memory storage or computationally intensive. Step 6 involves the use of GA operators to produce the next generation and is a prepatory step for Steps 2 - 5 in the next iteration. As Step 6 is stochastic (depending upon the probabilities of crossover and mutation) there is no fixed time for completion of this step (however, we can put a limit on the computations). The operations as explained in Section 2 are relatively simple and, in general, can be completed without excessive computation. Along with the sampling period considerations described above, the key to making the method computationally viable is the choice of liberal domain limits for the parameterization. One of the complexities of GA_{10} (that GA_2 also has) is that GA_{10} can produce strings outside the designer defined domain limits. For example, if we allow three digits to the left of the decimal point for K_p representation and choose domain limits as [-100,100] then eight out of 10 perturbations of the first digit will result in an unusable string. Limit checking and string regeneration functions are provided which also serve to place a limit on the number of crossovers/mutations allowed; however, these procedures introduce additional computation. The choice of liberal domain limits can forestall some of this complexity. Some further discussion of computational complexity issues is provided after the introduction of the GMRAC method.

3.1.3 Ship Steering Example

As an example, we apply GMBC to a cargo ship steering problem. The problem is to use GMBC to adaptively control the heading of the ship, ψ by manipulating the rudder angle, δ. A coordinate system fixed to the ship is shown in Figure 3 and the problem was taken from [18, pp. 355-359]. The cargo ship is described by a third order nonlinear equation [23, p. 27] and we use this in all simulations:

$$\dddot{\psi}(t) + \left(\frac{1}{\tau_1} + \frac{1}{\tau_2}\right)\ddot{\psi}(t) + \left(\frac{1}{\tau_1\tau_2}\right)H(\dot{\psi}(t)) = \frac{K}{\tau_1\tau_2}(\tau_3\dot{\delta}(t) + \delta(t)) \qquad (6)$$

where $H(\dot{\psi})$ is a nonlinear function of $\dot{\psi}(t)$. The function $H(\dot{\psi})$ can be evaluated from the relationship between δ and $\dot{\psi}$ at steady state such that $\dddot{\psi} = \ddot{\psi} = \dot{\delta} = 0$. An experiment known as the "spiral test" approximates $H(\dot{\psi})$ by $a\dot{\psi}^3 + b\dot{\psi}$. The real valued constants a and b are assigned a value of one for all simulations. The constants K, τ_1, τ_2, and τ_3 are defined as

$$K = K_0\frac{u}{l} \qquad (7)$$

$$\tau_i = \tau_{i0}\frac{l}{u} \quad i = 1, 2, 3. \qquad (8)$$

where u is the forward velocity of the ship in meters/sec and l is the length of

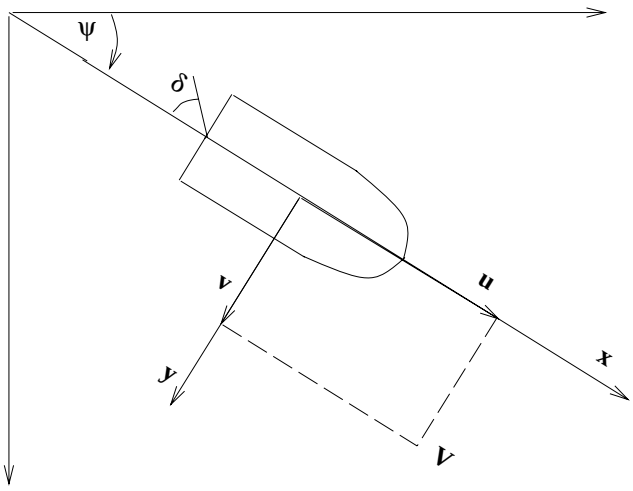

Figure 3: Cargo Ship

the ship in meters. For the cargo ship $K_0 = -3.86$, $\tau_{10} = 5.66$, $\tau_{20} = 0.38$, $\tau_{30} = 0.89$, $l = 161\,m$ and $\bar{u} = 5\,m/s$ (nominally).

The magnitude of the maximum allowable rudder angle is 80^o and $u(k)$ is normalized by this value (i.e., we will choose $p_3 = \frac{\hat{u}(k)}{80}$ in the pseudo-code for GMBC). The model, P, used by GMBC is the linear, zero-order hold, discrete equivalent ($T = 0.05$ seconds) of the following third order linear continuous time plant provided in [18]

$$G(s) = \frac{K(1 + s\tau_3)}{s(1 + s\tau_1)(1 + s\tau_2)} \tag{9}$$

where K, τ_i, τ_2, and τ_3 are defined as in equations 7 and 8. The PD type control law used is:

$$\delta(k) = K_p\psi_e(t) - K_d\dot{\psi}(t) \tag{10}$$

where $\psi_e(k) = \psi_r(k) - \psi(k)$, $\psi_r(k)$ is the desired heading of the ship.

For all simulations we choose $\psi_r(t) = 45^o$ for $0 \leq t < 250$ and $\psi_r(t) = 0^o$ for $250 \leq t \leq 500$. All initial conditions of the plant were set to zero at $t = 0.0$. The string length is a relatively minor factor in GMBC design and can be chosen depending upon desired accuracy. The total string length for K_p and K_d representation was 26 digits, 3 digits to the left and 9 to the right, plus 1 digit for sign information for each. The initial population of K_p and K_d gains was chosen by random number generation. After some experimentation, crossover and mutation probabilities were chosen as 0.82 and 0.33 with a population size of $n = 17$. Note that we use the relatively large mutation rate since the fitness function is time-varying and we must then ensure that the GA actively pursues

many possible regions in it search space (research in other on-line GAs has found the same thing; e.g., the work in [17] uses an even higher effective mutation rate via the use of "extinction and migration"). The search does not, however, degrade to a random walk in the parameter space. Note that since we use the elitism operator we are most often (unless we have a significant disturbance) guaranteed to have at least as good of a controller at time k as we did at time $k-1$. This has an overall effect of reducing the randomness of the search (especially considering the effect of the elite individual on selection). Elitism coupled with a slightly higher mutation rate has an overall effect of consistently providing a good controller yet making sure that the GAC is responsive to plant and environmental changes.

Figure 4 shows simulation results [3] if the values of α_1, α_2, and α_3 are 1.0, 1.0, and 0.003, respectively. Figure 5 shows simulation results for a different choice of α_i's, namely 950, 1.0, and 2.0 for α_1, α_2, and α_3, respectively. It is evident that by manipulation of the ratios of the α_i's we can control the response of the plant. Increasing the value of an α_i coefficient makes that term more important to the GA; however, it is the ratio between α_i's that is crucial in determining the response.

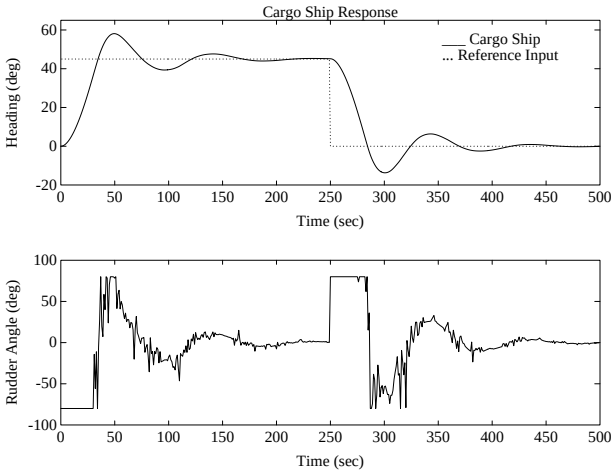

Figure 4: GMBC $\alpha_1 = 1.0$,$\alpha_2 = 1.0$,$\alpha_3 = 0.003$

The lower value of the α_3 coefficient (compare Figures 4 and 5) results in the GA using more control energy. This phenomenon makes sense because the use of control energy is not penalized as much compared to the second example. The GA is basically "free" to pick a large use of control over a small

[3] All simulations reflect data collected every five seconds. This number merely refers the frequency of data collection for the plots and is exclusive of system simulation. Therefore each figure with a time range of 500 seconds is a plot of 100 data points.

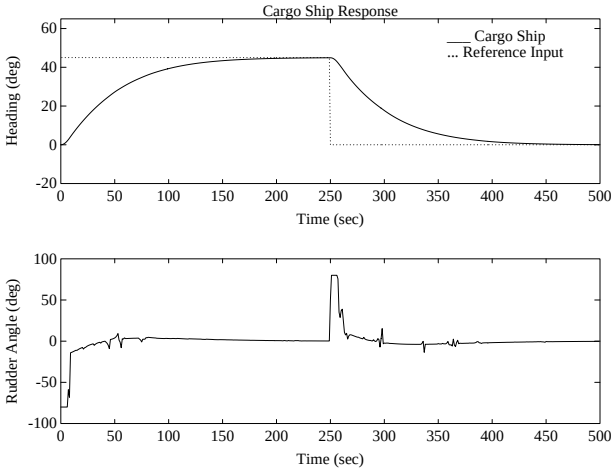

Figure 5: GMBC $\alpha_1 = 950.0$,$\alpha_2 = 1.0$,$\alpha_3 = 2$

use by keeping the value of α_3 small relative to the other coefficients. From the appearance of the graphs we also note that the same average behavior occurs over each interval of 250 seconds. This is hardly a trivial observation as the initial conditions at 0 and 250 seconds while comparable are not equal, and at 249 seconds the population of strings is locally tuned to reflect the current state of the plant (i.e., $\psi_e(t)$ and $\dot{\psi}(t)$ are small). As it has not been proven, this average behavior phenomenon is not guaranteed; however, for constant α_i's this behavior was always observed. It is suspected that despite the local tuning of K_p and K_d, a sufficient mutation probability can allow the GA to adapt and maintain acceptable system performance.

Figures 6 and 7 show the progression of fitness throughout the simulations of Figures 4 and 5 (note that zero fitness is best). As expected, the fitness goes to zero rapidly after a step input. In Figure 7 the fitness decreases dramatically at $t = 250$ seconds. Due to the increase in the α_1 coefficient in Figure 7, we are penalizing a change in error to a greater degree than in Figure 6. The graph in Figure 7 is a truncated version; the actual fitness decreased to approximately -1.9×10^6 at $t = 250.0$ seconds. The crossover and mutation rates are sufficient to drive the fitness to zero after this occurrence.

3.2 Genetic Model Reference Adaptive Control

The extension from GMBC to GMRAC is straightforward as it merely allows for the presence of a reference model to dictate how we wish the closed-loop system to respond to the reference signal. The objective function is similar to that of GMBC except that we make control decisions based upon the error between the

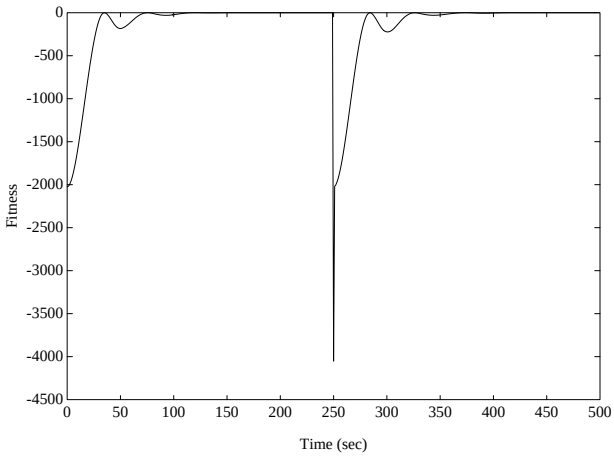

Figure 6: GMBC Fitness $\alpha_1 = 1.0$,$\alpha_2 = 1.0$,$\alpha_3 = 0.003$

actual system and the reference model. This change in the controller structure seems to dramatically increase the capability of the genetic adaptive scheme. GMRAC seems to be more resilient to model inaccuracies and disturbances than GMBC for the cargo ship studied. We will also show that GMRAC compares favorably to conventional techniques (gradient and Lyapunov model reference adaptive control (MRAC)) as well as fuzzy model reference learning control (FMRLC) for the cargo ship by comparing to the results in [23].

A general framework for GMRAC is shown in Figure 8. Note that in addition to the variables defined for GMBC we now have a reference model M which has an input $r(k)$ (or for predicting into the future $r(k) = \hat{r}(k+1)$) and an output $y_m(k+1)$. For the cargo ship the reference model M used is

$$\ddot{\psi}_m(t) + 0.1\dot{\psi}_m(t) + 0.0025\psi_m(t) = 0.0025\psi_r(t) \tag{11}$$

where ψ_m is the desired ship performance for the heading ψ_r. The reference model is taken from [23, p. 28] to reflect somewhat realistic performance requirements.

Similar to GMBC, the objective function utilized in GMRAC is of the form, $J_i = \sum_{j=1}^{N_p} \alpha_j {p_j}^2$. For this technique p_j might be the amount of error between the reference model and the plant output and again the α_j used as a scaling factor. For *selection* purposes, the value of J_i is mapped as in equation 5.
GMRAC proceeds according to the following pseudo-code assuming that the same PD controller adopted in GMBC is employed.

1. Initialize the GA. Choose the number of digits to represent each controller parameter K_p and K_d. Choose crossover probability p_c and mutation

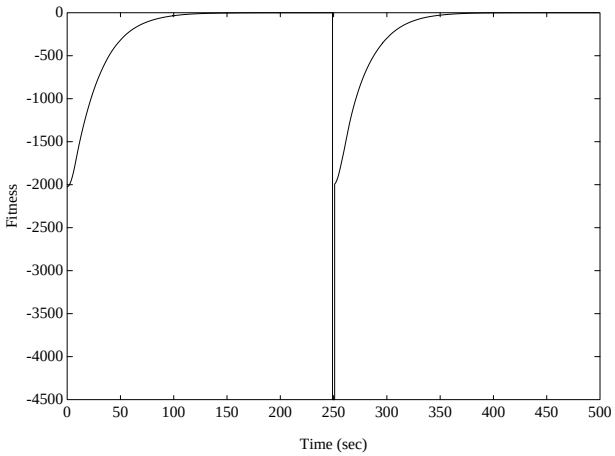

Figure 7: GMBC Fitness $\alpha_1 = 950.0$,$\alpha_2 = 1.0$,$\alpha_3 = 0.003$

probability p_m. Generate an initial population of K_p and K_d gains (we make a random selection). Initialize sample time, T. Set time, t, to zero. Set initial conditions for plant and model of plant P. We choose all initial conditions to be zero.

2. Collect $y(k)$ and $r(k)$.

3. Generate $\hat{u}(k)$, for each population member C_i, $i = 1, 2, \ldots, n$ using the PD control law $\hat{u}(k) = K_p e(k) - K_d e_{dt}(k)$ where $e(k) = r(k) - y(k)$ and $e_{dt}(k) = \frac{e(k)-e(k-1)}{T}$. Then generate $\hat{y}(k+1)$ using equations 3 and 4. Thus the architecture of the "operation module" can be modeled as in Figure 2 except that now $\hat{r}(k+1)$ is passed through M to produce $y_m(k+1)$. Note that as with GMBC $r(k) = \hat{r}(k+1)$.

4. Assign fitness to each element of the population C_i, $i = 1, 2, \ldots, n$:
   Let $p_1 = e_m(k) - e_m(k-1)$ where $e_m(k) = y_m(k) - y(k)$. Let $p_2 = y_m(k+1) - \hat{y}(k+1)$. Let $p_3 = \frac{\hat{u}(k+1)}{80}$ (recall that we scaled p_3 in a similar manner earlier).

   $$J_i = -(\alpha_1 p_1^2 + \alpha_2 p_2^2 + \alpha_3 p_3^2)$$

5. The maximally fit C_i becomes c^*. This c^* is the controller used between times k and $k+1$.

6. Produce the next generation using GA operators.

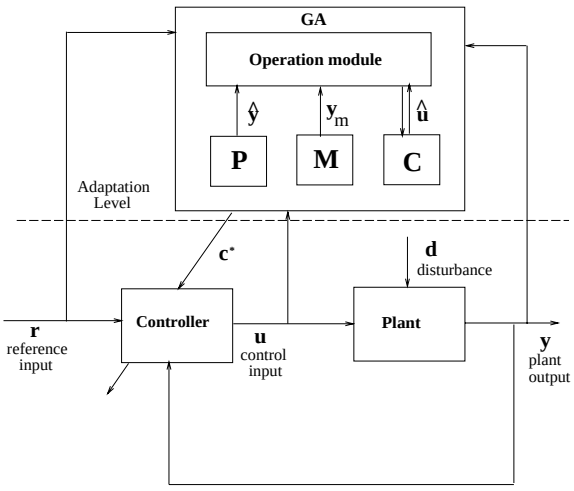

Figure 8: Genetic Model Reference Adaptive Control

7. Let $t := t + T$. Go to Step 2.

Note that the complexity of GMRAC is only slightly higher than that of GMBC. In fact, cursory investigations of this algorithm with respect to real-time implementation prove satisfactory. For example, for a population size of 19, string length of 20 and representative crossover and mutation probabilities (0.6 and 0.24, respectively) the calculation of one c^* (one generation, i.e., one discrete time step) requires approximately 0.02 seconds [4]. We obtained this value even though in its present form the program is not optimized and extraneous programming which facilitates analysis of the genetic adaptive technique is present. For a real-time implementation it is clear that the time to compute the control value would be even less.

Next, we establish design guidelines for GMRAC using the cargo ship as a case study. With regard to the choice of mutation and crossover probabilities and α_i's, we expect a similar amount of complexity in determining exact cause and effect relationships as for GMBC. However, guided by our investigations of GMBC in the previous section we can expect the following general features:

1. The ratios, $\frac{\alpha_3}{\alpha_1}$ and $\frac{\alpha_3}{\alpha_2}$, will be the most important factors in determining system response. Ratios can be scaled by the user to reflect the importance of either system error or use of control authority. If the ratios are large, we except the amount and rate of change of system error to be sacrificed at the expense of the use of small control energy. Smaller ratios can offer varying degrees of trade-off.

[4]This value is based upon averages over 4, 40, and 400 generations run on a Unix-based Sun4 Sparc station.

2. A high crossover probability promotes "learning" between generations and a strong base of "good" controllers can be built.

3. The mutation probability should be kept large enough to allow for adaptability but not so large as to completely change the "non-elite" members of the population with each generation.

Using the above trends as guidelines we present the following method for determining GMRAC parameters for the cargo ship steering example.

1. Fix string length, population size, and crossover and mutation probabilities. If one were to actually implement this control scheme then population size and string length would be bounded by the sampling rate. As discussed previously the crossover probability should be chosen to be relatively large (i.e., 0.6 to 0.8). We have found that a good initial guess at the starting mutation probability is to use approximately 40% of the chosen crossover probability (i.e., 0.24 to 0.32 – see our discussion in the last section on why this is a relatively high probability). For the cargo ship we choose a string length of 18 to code the parameters of the PD controller: 3 digits to the left of the decimal, 5 digits to the right and 1 digit to convey sign information for each parameter. The population size is set at 17. Crossover and mutation probabilities are chosen as 0.6 and 0.24, respectively. As mentioned in the first subsection of this section, the effect of crossover and mutation will in general be subordinate to the effect of the α_i's. Therefore we fix the probabilities and concentrate our design on the proper choice of the α_i's.

2. Choose an initial set of α_i's. Keep the value of the control energy coefficient small with respect to the other coefficients. For the PD adaptive scheme described above this means keeping the ratios, $\frac{\alpha_3}{\alpha_1}$ and $\frac{\alpha_3}{\alpha_2}$ small. A good starting set might be $\alpha_1 = 50$, $\alpha_2 = 500$, and $\alpha_3 = 0.001$. Figures 9 and 10 show system responses for this choice of controller parameters. Notice that the plant output tracks the reference model almost perfectly.

3. At this point, the criteria for an acceptable system response must be determined by the user in order to decide what parameters to tune. Looking at Figure 10 we see a large use of control energy and high frequency fluctuations in the error between the reference model and the cargo ship heading. Suppose the latter phenomenon is unacceptable in our design. The fluctuations are occurring because we are not penalizing the change in model error enough. Let us increase the value of the α_1 coefficient from 50 to 500. Figure 11 shows the response of the system. The cargo ship heading behaves as it did in Figure 9; hence we omit the plot. We notice that the model error changes in a smooth fashion, but perhaps the amount of control is still unacceptable.

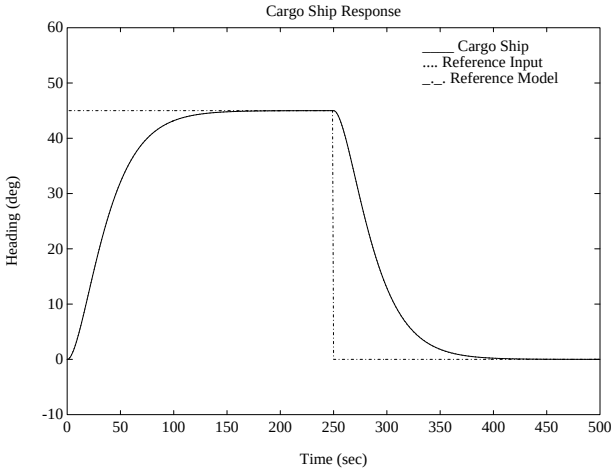

Figure 9: Cargo Ship Response for each set of α_i's

4. To alter the use of control energy, we can either increase the value of α_3 or decrease the value of α_2. Both changes represent an attempt to allow more system error by decreasing the control authority used. We choose the latter alternative and change the α_2 coefficient to 50. We obtain the response shown in Figure 12 where once again the heading response is is virtually identical to that of the two previous trials so we omit the plot. We have successfully reduced the large variations in the rudder angle and the high frequency fluctuations in the error between the reference model and cargo ship heading; however, the reference model error increased slightly (representing a typical trade-off).

The α_i values for GMRAC in Figure 12 provide better responses than the gradient and Lyapunov MRAC approaches in [23, pp. 31-32] for this same cargo ship steering application. Gradient and Lyapunov MRAC techniques show a large system overshoots until the K_p and K_d converge to provide a good tracking response and exhibit the similar amounts of control energy for the third GMRAC case (Figure 12). GMRAC provides a heading response that compares well with FMRLC in [23, p. 31].

Now that we have obtained an acceptable controller, we will investigate how the system responds to various disturbances. The effect of a disturbance input acting through the rudder angle characterizes the effect that wind and waves have on the control that is applied to the plant. Figure 13 shows the system response for a sinusoidal disturbance at the rudder of $1 + 2 * \sin(2\pi\, ft)$ where f is one cycle per minute. GMRAC proves to be quite robust to this type of disturbance as opposed to GMBC where the ship heading perceptibly

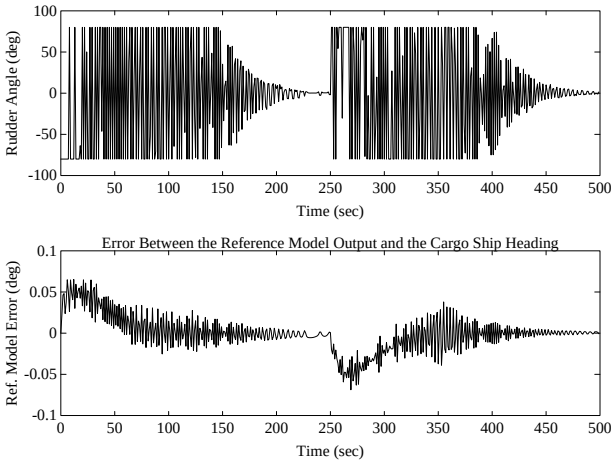

Figure 10: Cargo Ship Response: $\alpha_1 = 50$, $\alpha_2 = 500$, $\alpha_3 = 0.001$ (reference model error $= y_m - y$)

changed (we omit the plots in the interest of brevity). Furthermore, the GMRAC technique compares favorably to the disturbance response of the FMRLC as shown in [23, p. 32].

From equation 6 we see that the dynamics of the cargo ship are sensitive to the speed of the ship. Intuitively this makes sense because the effectiveness of the rudder decreases as speed decreases (so the ship is harder to control at lower velocities). Figures 14 and 15 show how the GMRAC will respond to a changing velocity profile. Clearly, the system responds well to this type of variation in the underlying ship dynamics. This is rather surprising considering that GMRAC bases its controller selection decisions upon a fixed model that has been obtained by *linearizing* the nonlinear plant using a velocity of $5\,m/s$ and the ship is harder to control at a velocity of $3\,m/s$.

What if we cannot make all of the necessary computations within an acceptable fraction of the sampling rate? Suppose GMRAC can only provide a c^* every 4 sampling periods and this controller is based upon information collected 4 sampling periods ago (in the interim the plant has been controlled by the previous c^*). Figure 16 shows the effect on the system of delay due to excessive computational time in the adaptation loop. We see that the error and amount of control increase but not excessively (compare to Figure 12).

Finally, suppose that there is an order mismatch between P and the plant defined by equation 6. Åström and Wittenmark [18, p. 356] propose a second order linear model for the cargo ship. If we use the second order linear model as P but still simulate the actual plant using the third order nonlinear differential equation, Figure 17 shows that we still obtain good results.

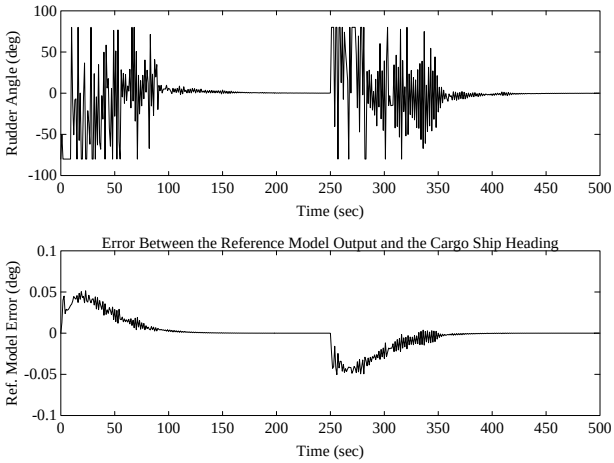

Figure 11: Cargo Ship Response: $\alpha_1 = 500$, $\alpha_2 = 500$, $\alpha_3 = 0.001$ (reference model error $= y_m - y$)

The reader may be concerned with all the random actions that are taken by the adaptive schemes. It is somehow a bit disconcerting, not quite as comfortable to believe in as opposed to conventional deterministic adaptive techniques. Looking at Figure 19 which plots the K_p and K_d for the GMRAC (with the response shown in Figure 18) it is not quiescent at all - it's a virtual "storm" of digits with seemingly no discernible pattern. If we take a simple average over the simulation we can calculate an average of K_p and K_d of -6.76 and -26.038, respectively. If we compare these values to the equilibrated values of the Lyapunov (-3.0974, -105.0242) and gradient (-4.775, -171.85) techniques in [23] this offers some reassurance that despite the randomness perhaps some reasonable average behavior is achieved.

4 Genetic Algorithm Based Supervisory Control

In this section and the next one we attempt to resolve two issues of GMRAC operation which were reviewed in simulations at the end of Section 3: (i) the choice of the scaling factors of the objective function (α_i's) and the choice of crossover/mutation probabilities and (ii) choice of the model P used to calculate $\hat{y}$. Instead of having the designer choose the α_i's and the model P, we study the use of a hierarchical scheme where the subordinate controller, GMRAC, is supervised by another GA that evolves these quantities. In the first subsection of this section we discuss a basic GA supervisory scheme. In the second subsection

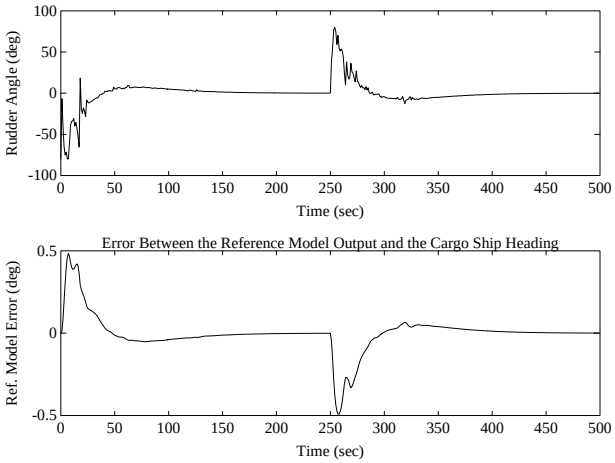

Figure 12: Cargo Ship Response: $\alpha_1 = 500$, $\alpha_2 = 50$, $\alpha_3 = 0.001$ (reference model error $= y_m - y$)

we augment this concept by introducing "population splitting" which allows for reduced computational complexity and faster adaptation. In Section 5 we show how to use GASC coupled with a GA-based estimation scheme that estimates P.

In GA supervision we extend the concept of design evolution to the objective function itself in an attempt to provide a "learning" system that can choose the best objective function for meeting the specifications represented in the reference model. To implement this concept, we will use a GA to make choices regarding the value of the α_i's (weights in the objective function) and crossover/mutation probabilities of the GMRAC system. The GA that makes these decisions will be called a Genetic Algorithm-based Supervisor and this type of control will be called GA-based Supervisory Control (GASC).

To propose GA control of another GA immediately conjures up questions of utility: Why add this complexity? We have already shown in Section 3 that with proper choice of the α_i's and crossover/mutation probabilities we obtain good performance with the GMRAC. If we suggest one level of GA supervision, why not have two or more levels of GASC (i.e., supervisors for supervising)? Where is the point where we obtain no benefit by increasing the system complexity? To answer some of these questions, we must first outline the supposed benefit of one level of GASC.

The purpose of the GA supervisor is to evolve sets of α_i's and crossover/mutation probabilities to arrive at the optimal sets which are most effective in reducing error and control energy used in the system. The general framework for GASC is shown in Figure 20. In Section 3 we chose one set of GMRAC parameters

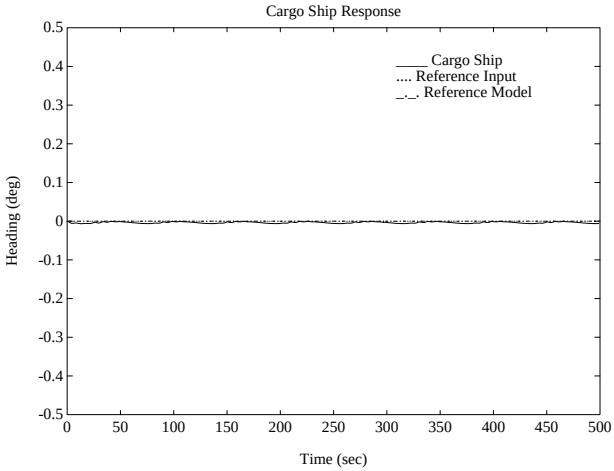

Figure 13: Response to Sinusoidal Disturbance at Rudder: $\alpha_1 = 500$, $\alpha_2 = 50$, $\alpha_3 = 0.001$

and tuned these to obtain good performance. We have no way of knowing if a different set of controller parameters would result in an even better performance (i.e., use of less control energy, smaller system error and, greater resilience to the studied disturbances). By adding the GA supervisor to GMRAC we are adding a method for seeking optimal GMRAC parameters. While the GMRAC structure of the combined GASC/GMRAC will still operate using generation-to-generation (local) optimization as discussed in Section 3, the GASC will guide the system towards improved performance, i.e., it will learn the best objective function to use for GMRAC. We will study adding just one level of supervisory control. Clearly, more levels could be added if there was sufficient justification (i.e., where the choice of scaling coefficients of the objective function of the GASC were difficult to determine without excessive trial and error). One level of supervision proved satisfactory for our needs.

GASC also offers two benefits in terms of control methodology. It provides the controller a way to "think" about how to control the system in terms of various "high" level concepts: (i) the summation of the error-squared between the reference model and the actual plant over a time window, (ii) the control energy used in the system over a time window, (iii) the maximum error-squared in the system over a time window, (iv) the size of that time window, and (v) any other parameters that can be perturbed to vary the performance of the system. GASC acts as a "tuner" for the control system. Because of GASC's use of "high" level concepts it has a large degree of flexibility in its application. While (v) above can be used to include problem specific information, the other concepts can be applied to virtually all systems.

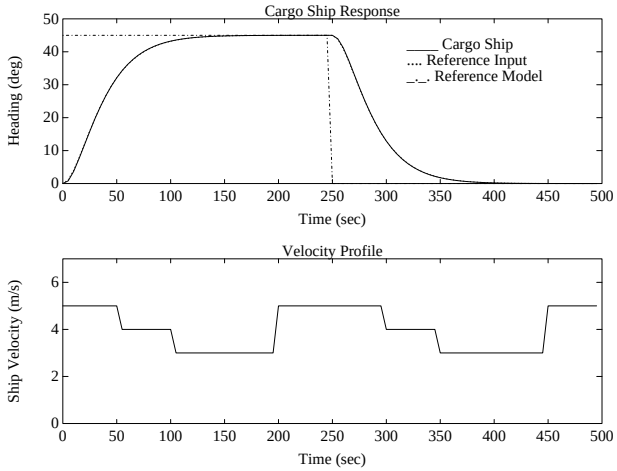

Figure 14: Changing Velocity Profile: $\alpha_1 = 500$, $\alpha_2 = 50$, $\alpha_3 = 0.001$

While the first benefit outlined above is typical of hierarchical control structures, a second benefit of GASC is more specific to genetic supervisory control. While some hierarchical control structures "learn" to control a system (as GASC does) it is just as important that GASC makes mistakes. Without the specification of some termination condition, GASC is constantly searching for areas of improved performance. It does not "know" *a priori* where these areas are; the GA operators give it a systematic method for searching, but as average/maximum fitness is not guaranteed to increase over any span of generations, it will also locate areas of bad performance (i.e., it makes mistakes). Normally, this penchant for making mistakes would not be desirable in a controller; however, in this learning system it can be viewed as a positive attribute. A mistake in the control of the system (i.e., a bad controller – one that if employed over all time would, e.g., result in an unstable system) for one particular set of operating conditions can be a benefit for a different set of operating conditions. Because of the stochastic nature of GASC, it has the ability to learn how to control a plant according to some specified level of performance while continually introducing new phenotypes into the system which could control the plant better in the future when it enters different regions of operation.

4.1 Basic GASC

GASC performs tuning upon receipt of information regarding the error between the reference model and the actual response and the amount of control energy used over the time window (See Figure 20). As with GMRAC we can describe the objective function of GASC as $B_i = \sum_{j=1}^{N_q} \beta_j q_j$ where q_j represents infor-

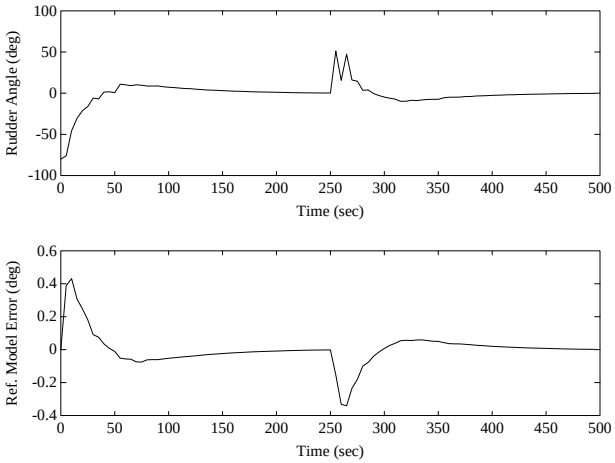

Figure 15: Changing Velocity Profile: $\alpha_1 = 500$, $\alpha_2 = 50$, $\alpha_3 = 0.001$ (reference model error $= y_m - y$)

mation garnered from GMRAC over a specified time period, T_s. For example, q_j might represent the summation of the error-squared over a moving window of length N_t samples (i.e., the past N_t samples) which we denote as $q_j = \sum_{N_t} e(k)^2$. The principle is still similar to that of GMBC and GMRAC, but now, choice of the β_i's determines how GASC will tune GMRAC. Population members of GASC are coded in terms of parameters of GMRAC, namely the α_i's, crossover and mutation probabilities, and any problem specific information we might wish to include. For example, in the application of GASC to the cargo ship steering problem we allow the value of the maximum allowable rudder angle to be a tuned quantity (i.e., in Section 3 the maximum allowable rudder angle was 80^o; here we allow this controller parameter to vary). The essential goal of GASC is to accurately control the plant so that the closed-loop system behaves like the reference model using the least amount of control energy.

For the cargo ship steering application GASC proceeds according to the following pseudo-code:

1. Initialize GMRAC as in Step 1 of the GMRAC pseudo-code in Section 3.

2. Initialize GASC. Choose string length, p_c, and p_m for GASC. Generate an initial population of GMRAC parameters with elements s_i via, e.g., random number generation. $s_1 = \alpha_1$, $s_2 = \alpha_2$, $s_3 = \alpha_3$, $s_4 = p_c$, $s_5 = p_m$, $s_6 = u_{max}$ ($u_{max} = 80$ in Section 3).
   Let the ith GASC population member, S_i be one string consisting of the concatenation, $s_1 s_2 s_3 s_4 s_5 s_6$.

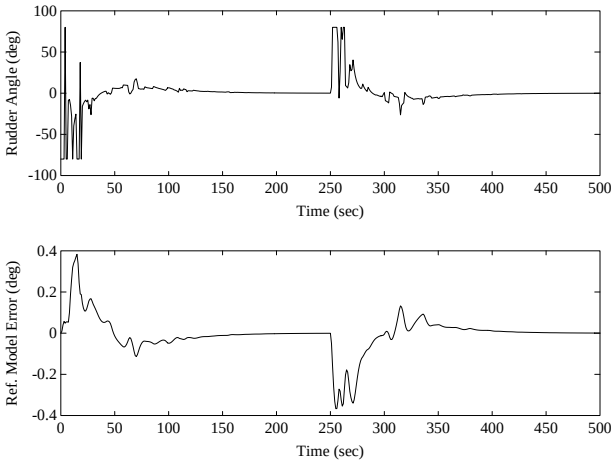

Figure 16: Computational Delay: $\alpha_1 = 500$, $\alpha_2 = 50$, $\alpha_3 = 0.001$ (reference model error $= y_m - y$)

Set time, t, to zero. Let $i := 1$, so that we are processing S_1, the first string in the population.

3. Start GMRAC and let it operate until $t + T_s$ with parameters specified by S_i.

4. Let $t := t + T_s$.

5. Collect GMRAC data , q_j, $j = 1, 2, \ldots, N_q$ for the time period, $T_d = \{k, (k-1), \ldots, (k-N_t)\}$. $N_t = \frac{T_s}{T}$ is an integer, where T is the sampling period of the plant and T_s is the window of time over which we will evaluate each q_j. Suppose that $N_q = 4$ and that

   $q_1 = \sum_{k \in T_d} \psi_e(k)^2$ where $\psi_e(k) = \psi_r(k) - \psi(k)$.

   $q_2 = \sum_{k \in T_d} \left(\frac{u(k)}{u_{max}} \right)^2$.

   $q_3 = \max_{k \in T_d} \{e(k)^2\}$.

   $q_4 = (\psi_e(k)^2|_{k \in max\, T_d} - \psi_e(k)^2|_{k \in min\, T_d})^2$

   Thus q_1 represents the error-squared sum, q_2 is the normalized control energy sum, q_3 is the maximum error-squared, and q_4 is the change in the error-squared over the time interval T_d.

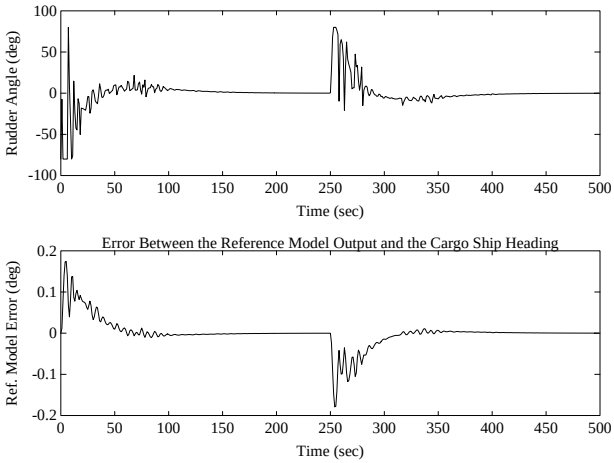

Figure 17: Model Mismatch: $\alpha_1 = 500$, $\alpha_2 = 50$, $\alpha_3 = 0.001$ (reference model error $= y_m - y$)

6. Assign fitness to S_i:

   $G_i = \sum_{j=1}^{N_q} \beta_j q_j$. Map $-G_i$ to a positive fitness as in equation 5.

7. If all members of the population have not been processed (assigned a fitness value) let $i := i + 1$. That is, prepare to evaluate the next string in the population over the next T_s seconds. Go to Step 3. When all members of the population have been processed, produce the next generation using GA operators (as with GMBC and GMRAC). Let $i = 1$. Go to step 3.

To illustrate the use of GASC we will implement this technique on the cargo ship steering application. Intervals for components of the S_i were chosen from our experience using GMRAC for the cargo ship steering problem and are assigned as follows:

s_1: [1,3000]
s_2: [1,2000]
s_3: $[1 \times 10^{-8}, 0.009]$
s_4: [0.5,1.0]
s_5: [0.0,0.5]
s_6: [40,80]

After some simulation based investigations the GASC objective function gains were chosen to be $\beta_1 = 2000$, $\beta_2 = 1$, $\beta_3 = 1000$, and $\beta_4 = 4500$. The total

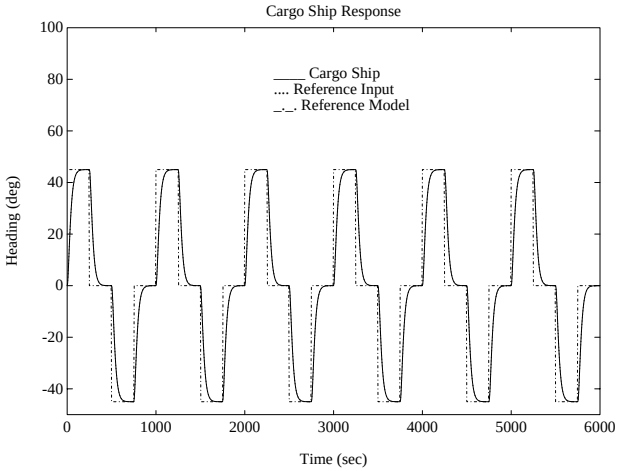

Figure 18: GMRAC response

population size was 40 with each string having a length of 51 digits. The value of T_s was 5.0 seconds. Crossover and mutation probabilities for GASC were chosen as 0.82 and 0.025, respectively. Notice that we have chosen the mutation probability to be much smaller than in the case for GMRAC. We made this choice because we view the GACS as a global optimizer for GMRAC so that its parameters should be chosen similar to those of a GA used to solve an off-line design problem.

All values were obtained by experimentation with the system but the rationale for the choice of s_i limits and the β_i's is similar to the reasoning proffered for GMBC and GMRAC. Interactions are complicated by the fact that the underlying GMRAC which is being tuned by GASC is stochastic. We notice that the maximum ratio between s_3 and s_1 or s_2 is 0.009. This is in keeping with the guidelines illustrated in Section 3. Limits for s_4 and s_5 were chosen to effectively span the range of reasonable probabilities. Examination of the β_i's shows that the change in modeling error over the time period is heavily weighted thus we expect adaptation to be relatively slow. Despite the values of the β_i's it is also necessary to look at the value of the quantities over which the β_i's will be acting. Typically, the value of q_1 will be the largest as it is the sum of the error over a time period. The value of q_2 can be no larger than the total number of samples within the time window. The value of q_3 will be a small fraction of q_1 and q_4 will also in general be small since it is only the difference between the starting error-squared and ending error-squared. Thus GASC is most likely to tune the system by choosing GMRAC parameters which are most effective in (first) reducing the amount of system error, (second) reducing the amount of control and (third) reducing the change in model error. It is difficult to deter-

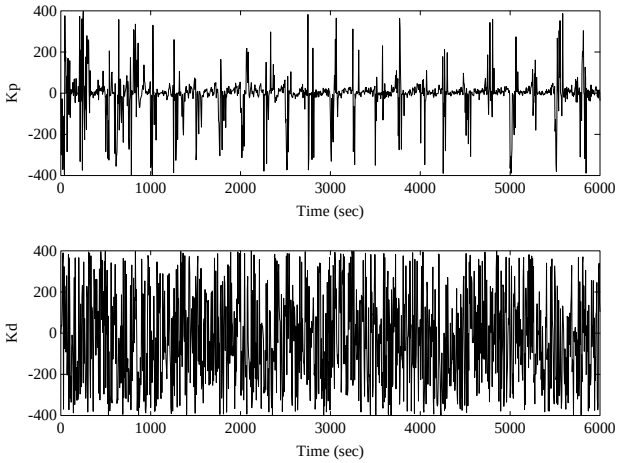

Figure 19: K_p and K_d gains for the GMRAC

mine the effect of the scaling on the maximum error-squared, but it is included in an effort to enhance the best fitness search. The ratios $\frac{\beta_2}{\beta_1}$ and $\frac{\beta_2}{\beta_4}$ are small so that the amount of error in the system is not completely sacrificed to the amount of control energy used.

Figures 21 and 22 show the cargo ship heading for the supervised system. Clearly, GASC is reducing the amount of system error while attempting to maintain and/or minimize the use of control. System adaptation is relatively slow, thus system error is reduced in a gradual manner. If desired the β_j's could be adjusted so as to give highest priority to the reduction of control energy.

4.2 GASC with Population Splitting

Another method of supervision involves partitioning the main population of the GASC into multiple sub-populations. In our population splitting method there is no interaction between sub-populations and therefore this splitting is nothing more than running two or more GA's at the same time. GA operators, *selection, crossover, mutation, and elitism*, proceed as described previously but only occur between strings within the same population split. The criteria for population splitting is specified by the user and should relate in a substantive manner to the performance of the system. For instance, population splitting may allow the GA to learn to control different operating regions of a plant without perturbing the entire population to do so. Situations where a plant is highly nonlinear might call for radically different control strategies in different operating regions. If the population is not split, GASC/GMRAC may not be able to adapt quickly enough to stabilize a particular region's dynamics. Secondly, the processing

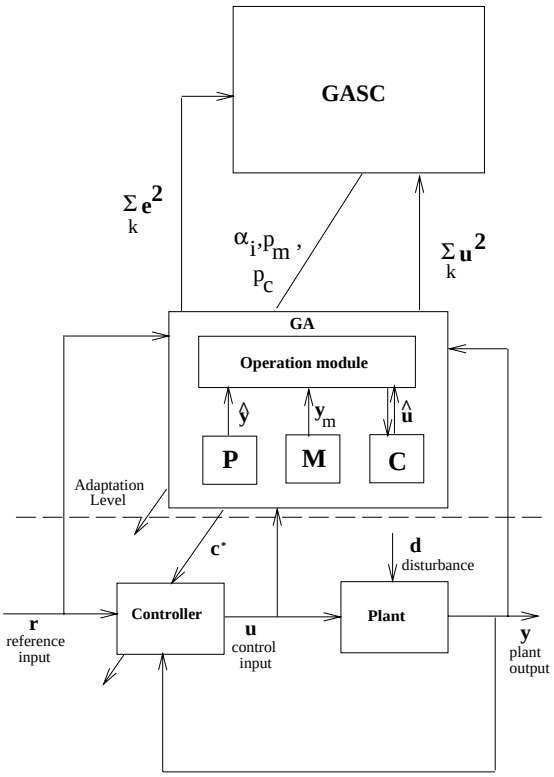

Figure 20: GA-based Supervisory Control (GASC)

time to obtain GMRAC parameters is substantially decreased (i.e., it is less computationally intensive if GASC only has to process 15 strings at a time as opposed to 40 strings).

To allow for the same rate of chromosome update between sub-populations, the mutation probability specified by the user is accorded to the one of the sub-populations (designer-specified) and the mutation probabilities of the other sub-populations are adjusted to match this mutation rate. For example, a total population of 40 members is divided into two sets of 25 and 15, respectively, with a mutation probability of 0.25. The string length of each member is 40 digits. The sub-population of 15 is assigned a $p_m = 0.25$ giving it a mutation rate of $15 \cdot 40 \cdot 0.25 = 150$ digits per generation. For equality we assign the sub-population of size 25 to have a p_m of 0.15. While it is of course possible, we do not perform the same type of equalization with respect to crossover rate since in some ways this operator is a measure of how often we want to combine "good ideas" of how to control the plant as opposed to mutation where we are not sure if the ideas introduced will be good or not. The equalization of

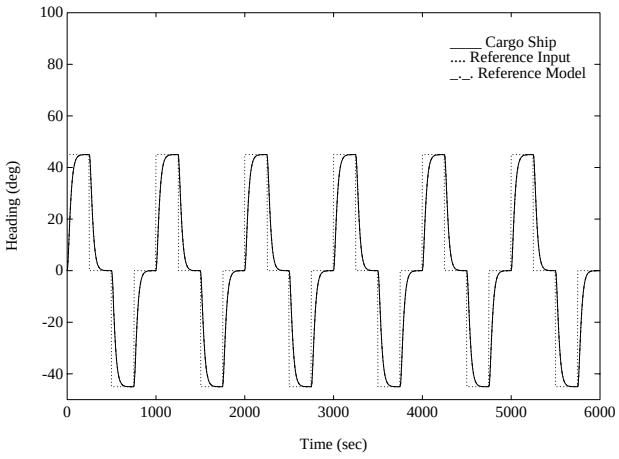

Figure 21: GASC: Cargo ship heading

mutation rate allows for the same degree of "randomness" to take place within each sub-population.

The elitism operator is used as discussed previously, however, we place a limit on the number of copies of the best fit string which propagates to the next generation. This prevents a "super-individual" from completely dominating the sub-population, making it incapable of learning new ideas due to premature convergence.

The crossover probability of the GASC was set at 0.8. The mutation probability, 0.020, was attached to sub-population P_1 and the mutation probability of P_2 adjusted for equality. Let $\bar{\psi}_e(k) = \left|\frac{\psi_e(k)-\psi_e(k-1)}{T}\right|$. We will use population P_1 at time k if $0 \leq \bar{\psi}_e(k) \leq \gamma$ and P_2 if $\bar{\psi}_e(k) > \gamma$, where $\gamma > 0$. The population sizes of the sub-populations are based upon the value of the change in reference model error in the system and the number of members in each sub-population will change every time a new run is initiated. For an overall population size of 40, typically, the number of strings in P_1 was approximately 30. It would also make sense to split populations based upon the value of $\psi_e(k)$ but it was found that the use of $\bar{\psi}_e(k)$ showed better performance. The explicit value of γ is chosen after initial processing of the entire population. For example, with a population size of 40 and a time period T_s, the value of γ is not calculated until $40 \cdot T_s$ seconds (i.e., all 40 strings have been processed). After $40 \cdot T_s$ seconds, the sizes of P_1 and P_2 are fixed. GASC with population splitting proceeds according to the same pseudo-code as outlined above except that there is an initialization phase where the entire population is processed (i.e., steps 1 - 5). Each string is assigned a value of $\bar{\psi}_e(k)$. As we are still evaluating data over time window T_s,

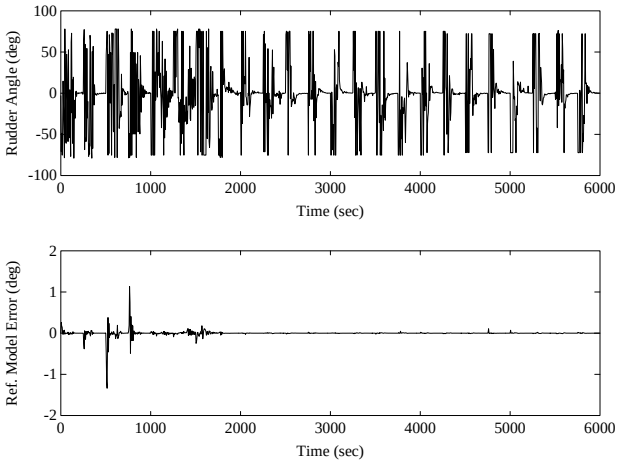

Figure 22: GASC: System error and control input (reference model error = $y_m - y$)

the value of $\bar{\psi}_e(k)$ at end of the time interval, T_s, is used to determine whether data from that interval is of P_1 or P_2. Once all strings have been processed γ is assigned to be one-third of the maximum value of $\bar{\psi}_e(k)$. Thus, strings S_i with values of $\bar{\psi}_e(k)$ less than or equal to γ form P_1 ("small") and those larger than γ form P_2 ("large"). Subsequent data from GMRAC is assigned a "small" or "large" label according to the value of $\bar{\psi}_e(k)$ at the end of the time interval and is processed in either P_1 or P_2, respectively.

For the cargo ship steering application[5], Figure 23 shows the GASC response for the case of population splitting where we have two sub-populations, P_1 and P_2. The cargo ship heading is the same as in Figure 21 and is not repeated. In Figure 22 the summation of the error-squared over the entire time interval is 7.2 as opposed to Figure 23 where the summation of the error-squared is 3.1. The summation of the control energy for GASC without population splitting is 1.57×10^6 as compared to 1.29×10^6 with population splitting. For this example, (GASC is a stochastic system) GASC has better performance (in terms of total system error) with population splitting. As it has a smaller number of strings to process for each sub-population, adaptation is quicker and less computationally expensive.

With respect to plant parameter variations (i.e., a velocity change in the cargo ship steering problem) GASC provides good performance for a wide range of velocities and still reduces system error and is moderately successful in reducing the use of control energy. Figures 24 and 25 show the system responses and

[5]We have also shown in simulation that the GASC with population splitting produces good results for an inverted pendulum control problem.

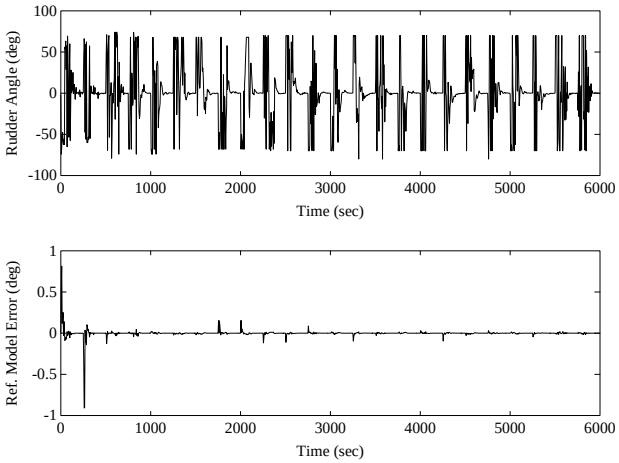

Figure 23: GASC: System error and control input with population splitting (reference model error = $y_m - y$)

use of control for the velocity profile in Figure 14. After some simulation based investigations the GASC objective function gains were chosen to be $\beta_1 = 9000$, $\beta_2 = 5$, $\beta_3 = 1000$, and $\beta_4 = 120000$. The total population size was 40 with each string having a length of 51 digits. The value of T_s was 5.0 seconds. Crossover and mutation probabilities for GASC were chosen as 0.82 and 0.025, respectively. With proper tuning of the GASC objective function gains we are able to obtain a response in Figure 25 which performs better than Figure 15 (assuming the same average behavior over 6000 seconds which we have validated in simulation) in terms of the amount of system error.

5 Genetic Algorithm Based Estimation for Adaptive Control

In this section we illustrate use of the GA in an on-line scheme to identify the plant model P that is used by GMRAC in making its fitness evaluation. Basically, in this section we combine many techniques outlined throughout this paper. We wish to find a collection of minimal order plant models which accurately represent the plant in different operating regions. To do this we will apply the normal GA operators *selection, crossover, mutation* (Section 2), and elitism (Section 3), and population splitting (Section 4), and the use of a "transmission digit", to be explained below, as a means of finding a minimal order representation.

For GA-based estimation (GAE), we might use sub-populations based upon

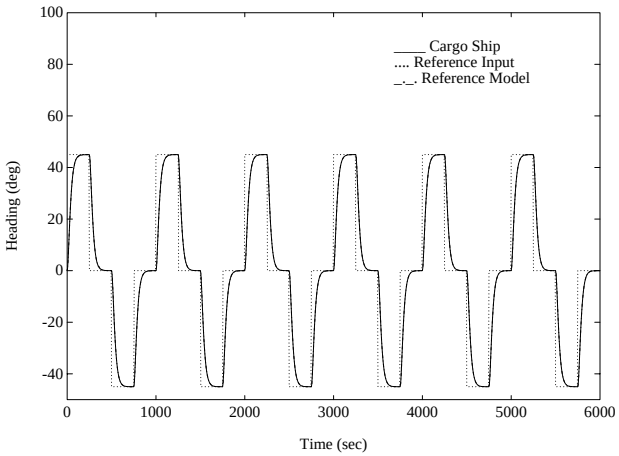

Figure 24: GASC: Cargo ship heading for changing velocities

the operating conditions of the plant. If we can accurately subdivide the populations to encompass the nonlinearity then a collection of linear models will be developed which describe the plant in different operating regions. These linear models can then be used to calculate $\hat{y}(k+1)$ in Figure 20. For the cargo ship example we construct three sub-populations (thus use three linear models) based upon whether the value of $|\bar{\psi}| = \left|\frac{\psi(k)-\psi(k-1)}{T}\right|$ is "small", "medium", or "large" (to be defined below) . The size of $\bar{\psi}$ affects the influence of the nonlinearity.

GAE has population members which represent the coefficients of a discrete-time representation of a strictly proper plant

$$\frac{\hat{y}(z)}{\hat{u}(z)} = K_z \frac{(a_{\bar{n}-1}z^{\bar{n}-1} + \cdots + a_0)}{(b_{\bar{n}}z^{\bar{n}} + b_{\bar{n}-1}z^{\bar{n}-1} + \cdots + b_0)} \tag{12}$$

where the compensator gain $K_z \in \mathcal{R}$, $a_j, b_k \in \mathcal{R}$, $j = 0, 1, \ldots, \bar{n}-1, k = 0, 1, \ldots, \bar{n}$ ($\hat{y}(z)$ and $\hat{u}(z)$ represent the z-transforms of $\hat{y}(k)$ and $\hat{u}(k)$).

The use of a "transmission digit" allows any of the a_j or b_k parameters to have a value of exactly zero or any value within the chosen domain. This "transmission digit" is merely an extra digit (having a 0 or 1 value) attached to each parameter which allows the parameter to decoded as zero regardless of the value of its digits or as its actual value. In this manner we can easily zero out any of the plant coefficients in our attempt to search for a minimal order representation. We allow any of the a_j or b_k parameters to be zero (subject to the caveat that we have a strictly proper, causal model). As we will only be using P to calculate the value of $\hat{y}$ one step into the future, the amount of accuracy required from the estimator can be somewhat relaxed and it is

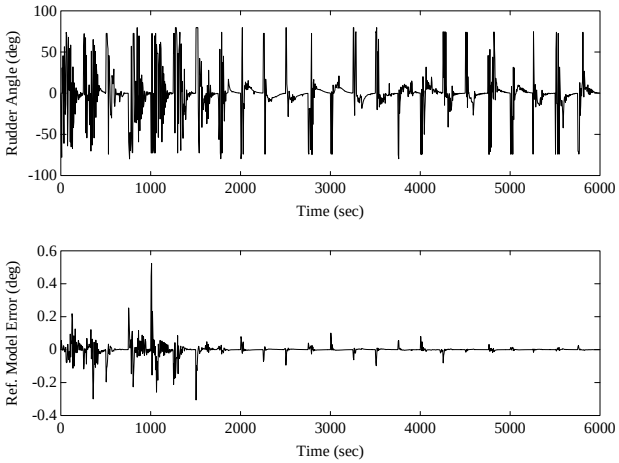

Figure 25: GASC: System error and control input for changing velocities (reference model error = $y_m - y$)

acceptable to look for a minimal order representation. For our implementation, proper estimation requires that we know the sign on the gain of the plant as it is possible for the GA to find linear models that result in controls of the wrong sign to be sent to the plant even though the input/output data is matched (similar conditions are needed in conventional adaptive control [18, 19]). Strings whose parameters are not of the same sign as the plant dc gain are rejected as possible candidates. In addition, strings which do not reflect a strictly proper plant are also rejected. For a proper plant, calculation of $\hat{y}(k+1)$ would require that we know or can estimate the value of $\hat{u}(k+1)$. Strings are selected for GA operators by $E_i = \sum_{k \in T_e} \left(\hat{y}(k) - y(k)\right)^2$ where $T_e := \{k, (k-1), \ldots, (k-N_e)\}$. The value of $-E_i$ is mapped to a positive fitness value as in equation 5. GAE determines an estimate for the plant model by selecting the set of parameters in equation 12 which minimize the error between the actual value of $y(k)$ and the estimate, $\hat{y}(k)$, for N_e samples of input-output data (hence our approach is similar to the off-line technique in [15]).

GAE implementation takes place entirely within the P model in Figure 20. Pictorially, we can think of GAE as in Figure 26 with the following variables defined:

Y Vector of N_e plant outputs,
$Y = [y(k), y(k-1), \ldots, y(k-N_e)]$

U Corresponding vector of N_e plant inputs,
$U = [u(k), u(k-1), \ldots, u(k-N_e)]$

$\hat{Y}$ Vector of estimated plant outputs,
$\hat{Y} = [\hat{y}(k), \hat{y}(k-1), \ldots, \hat{y}(k - N_e + \bar{n} + 1)]$
where $\bar{n}$ is as defined in equation 12.

P^* Best linear approximation of plant (i.e., the one with the best fitness)

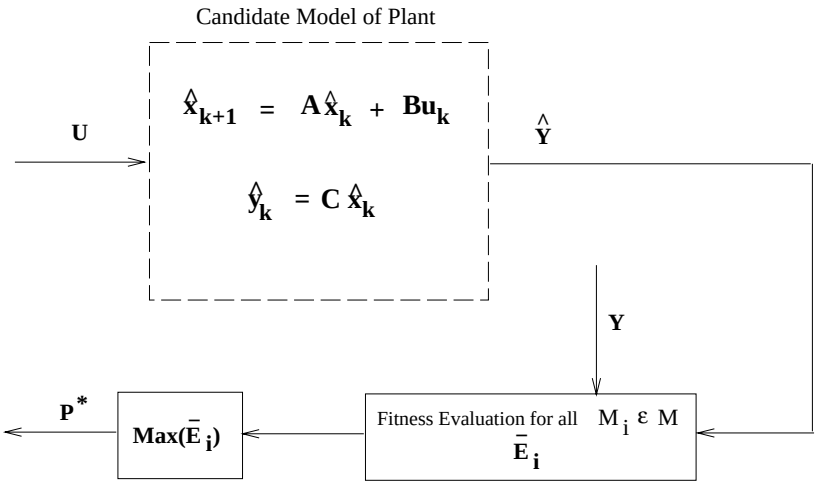

Figure 26: Genetic Algorithm-Based Estimator

GAE proceeds according to the following pseudo-code:

1. Choose string length, p_c, and p_m for GAE. Initialize the K_z, a_j and b_k in equation 12 via, e.g., random number generation. The ith string in the population is denoted by M_i.
2. Collect N_e samples of y and u data.
3. Calculate a set of $\hat{y}$'s , for each population member M_i using the vectors Y, U, and equation 12.
4. Assign fitness to each M_i: $E_i = -\sum_{k \in T_e} (y(k) - \hat{y}(k))^2$
5. The maximally fit M_i becomes m^*. This m^* represents the coefficients for the best approximation of P that we denote by P^*.
6. If the m^* calculated in this generation has a larger fitness than that of the previous generation's (denoted by m^*_{-1}) replace m^*_{-1} with m^*. If the m^* calculated in this generation has a fitness less than or equal to that of the previous generation's (denoted by m^*_{-1}) retain m^*_{-1} as the best estimate.

7. Produce the next generation using GA operators. Go to Step 2.

We will see in the system responses that the interaction between GAE and GMRAC is rather complex and it is postulated that a constant set of GMRAC parameters would not result in suitable performance. Therefore, we will use GASC to supervise the GMRAC/GAE system.

For GASC we define the following intervals for the parameters s_i $s_1 = \alpha_1$, $s_2 = \alpha_2$, $s_3 = \alpha_3$, $s_4 = p_c$, $s_5 = p_m$, $s_6 = u_{max}$ ($u_{max} = 80$ in Section 3):

s_1: [1,3000]
s_2: [1,2000]
s_3: $[1 \times 10^{-8}$,9.0]
s_4: [0.5,0.9]
s_5: [0.0,0.4]
s_6: [40,80]

GASC crossover/mutation probabilities were set at 0.65 and 0.02, respectively. The population size was 40 with a string length of 45. Sub-populations P_1 and P_2 are employed as outlined in GASC with population splitting. T_d for GASC was set at 5 seconds. The GASC objective function gains were $\beta_1 = 2000$, $\beta_2 = 1$, $\beta_3 = 0$, and $\beta_4 = 5500$. We remove the scaling on the maximum error-squared component to reduce system complexity. In addition, we allow calculation of the fitness (Step 3 of the GASC pseudo-code) only when GAE is providing accurate estimates of the plant so that data collected during the T_s interval is "valid" data. Very simply this means that GASC won't supervise a system it knows nothing about (if the P^* model is inaccurate, how can the GASC draw any conclusions about the performance of the underlying controller, GMRAC).

Naturally, GAE proceeds on an interval basis. For the cargo ship, which is sampled at 0.05 seconds, we initially "update" the estimator every 0.5 seconds based upon the previous 10 samples (or 0.5 seconds) of input/output data. "Update" is enclosed in quotes because it is quite possible that a particular generation's estimation of P^* will not perform as well as its predecessor. In this case, we can see from Step 6 above that the current P model is not changed. It can occur that the P^* model calculated for the case where $|\bar{\psi}|$ is "large" might perform better (i.e., provide more accurate estimates) than the P^* model calculated for the case where $|\bar{\psi}|$ is "medium" despite the actual value of $|\bar{\psi}|$ being "large". To compensate for this occurrence, every T_c seconds we allow the string with maximum fitness over the total population to be copied into the two sub-populations which do not carry this chromosome in their midst. The value of T_c for our simulations was 5 seconds.

For initialization of the estimator, we allow K_z to be any value between 0.99 and -0.99. Each a_j and b_k in equation 12 can have any values from -

10.0 to 10.0 ($\bar{n} = 2$ in our examples, so we are searching over second order models). Crossover and mutation probabilities are set at 0.80 and 0.09 with a total population size of 120 members. There are three sub-populations as described previously each of which contains 40 members. The total population is split according to the value of $|\bar{\psi}|$. $|\bar{\psi}| < 0.07$ is "small", $0.07 \leq |\bar{\psi}| \leq 0.6$ is "medium", and $|\bar{\psi}| > 0.6$ is "large". These splits were determined after several simulation-based investigations.

For the GMRAC candidate K_p and K_d gains the population size was 30 with a string length of 16. To the GMRAC objective function outlined in Section 3, we make two changes in Step 4.

1. Assign fitness to each C_i:

   Let $p_1 = e(k) - e(k-1)$, $p_2 = y_m(k+1) - \hat{y}(k+1)$, $p_3 = \frac{\hat{u}(k+1)}{80}$, and $p_4 = \hat{y}(k+1) - y(k)$.
   If ${p_4}^2 \leq \gamma$, then $E_i = -(\alpha_1 p_1^2 + \alpha_2 p_2^2 + \alpha_3 p_3^2)$ else $E_i = -(\alpha_3 p_4^2 p_3^2)$

Thus when our estimator is not doing a "good" job (i.e, $p_4^2 \geq \gamma$), we base the control decision only upon the value of $\hat{u}(k+1)$. This tends to minimize the control signal to the plant. When the estimator is providing accurate values, the usual objective function is used. The value of γ for the simulations was 10.0.

Figures 27 - 29 show the responses for the system. In initial stages of operation (i.e., the first 40 seconds. in Figure 30) the actual system response lags greatly behind the reference model response and the system generates large $\bar{\psi}$'s (by using large control signals) to reduce the system error. Secondly, once the reference model has reached the setpoint (45^o at approximately 110 seconds), we would like the controller to send a small control signal to the plant. This is impossible, however, because at this point $\bar{\psi}$ is "large" and a small control signal would result in a large error. Thus even when $\bar{\psi}_e$ is small the controller is still sending out large control signals. In this instance $|\bar{\psi}|$ is not small and no plant estimation is taking place for the "small" sub-population or if it is, the P^* model developed is only based upon large control signals. A second problem occurs when once $\bar{\psi}$ has been reduced to a small level and $\bar{\psi}_e$ is small, the small value of the control signal that the controller would like to send to the plant results in an inaccurate estimate from the P^* model thus decreasing its fitness with respect to a larger signal which gives a more accurate estimate. Certainly, we have a persistency of excitation [19] issue here that is complicated by the fact that the system that we are trying to identify is under closed-loop control. We need to perturb the system enough to discover the internal dynamics, but we are also trying to track a reference model while doing this. These two requirements can sometimes conflict. So, it is apparent that this is a multi-faceted problem for the combined GASC/GMRAC/GAE: (i) in the initial stages, we would like to relax the scaling on the error term (α_2) or perhaps increase the

scaling on the change in error term (α_2), allowing the system to expend less control energy and thereby decrease $\bar{\psi}$, (ii) we also wish to have persistency of excitation so that GAE can produce accurate models (in some sense $\bar{\psi}$ must be "large" for this to occur), and (iii) in later stages, we wish to reduce system error while also reducing the use of control energy. With the proper choice of GASC objective function gains and "good" set of initial GMRAC parameters, the combined GASC/GMRAC/GAE can perform adequately as seen Figures 27 - 29. With this implementation, however, it is difficult, to make a choice of either these gains or a "good" initial population.

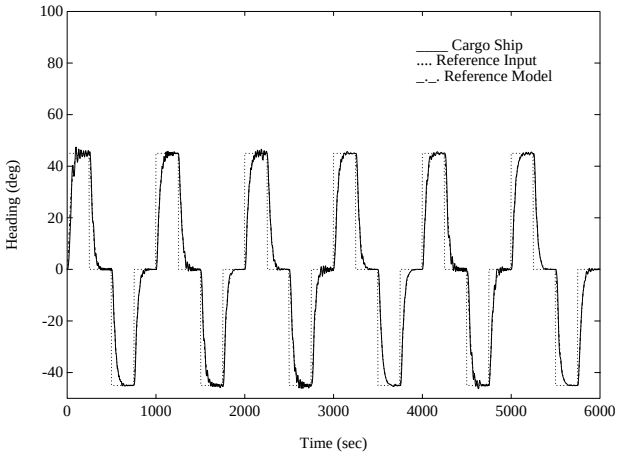

Figure 27: GAE: Cargo Ship

As a final note it is important to point out that while the GAE does not enhance the performance beyond that achieved with GASC and GMRAC alone (even after extensive tuning), this is not too surprising since GAE tries to automatically specify a model of the plant where in the past techniques it was assumed available. Actually, the combined GASC/GMRAC/GAE approach is very general. It tries to identify the plant it is trying to control and simultaneously make it act as the reference model specifies it should behave. This fundamental conflict between identification and control objectives that we see above with the combined GASC/GMRAC/GAE seems to exist in many adaptive control schemes [18, 19].

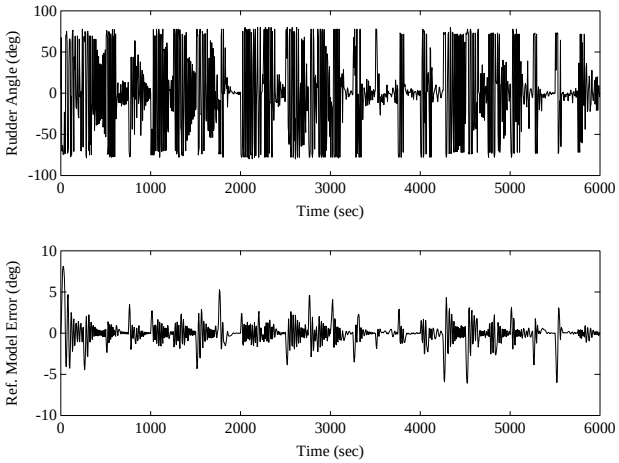

Figure 28: GAE: System error and control input (reference model error $= y_m - y$)

6 Concluding Remarks

We have shown that the genetic algorithm has a great deal of flexibility in its application to control systems. In particular, we introduced (i) two new (but closely related) approaches to genetic adaptive control, GMBC and GMRAC, and (ii) we introduced the idea of GASC, GASC with population splitting and GAE for use in GMRAC. While both GMBC and GMRAC depend on the accuracy of the model of the plant P that they use, GMRAC appears to be more resilient to various system disturbances. If an accurate P model is available, GMBC and GMRAC are relatively easy techniques to apply as tuning of the α_i's in the objective function can usually be approached in a methodical manner. In addition, GMBC and GMRAC are not very computationally complex and should prove amenable to real-time implementation. If the user does not wish to manually tune the GMRAC system, GASC offers an alternative which can provide an even better performance than that of a constant set of GMRAC parameters. GASC's hierarchical view allows the user to think of the control problem at a higher level and apply this manner of thinking to on-line tuning of GMRAC parameters. GASC with population splitting is an enhancement which allows for faster adaptation and decreased computation time. If no P model is available GAE may prove capable of delivering an accurate model; however, the combined GASC/GMRAC/GAE system seems to be difficult to tune and at this stage this is not an approach to embark on cavalierly.

While each of the investigations provides new ideas about how to use GA's to solve control problems, several topics for other investigations immediately suggest themselves:

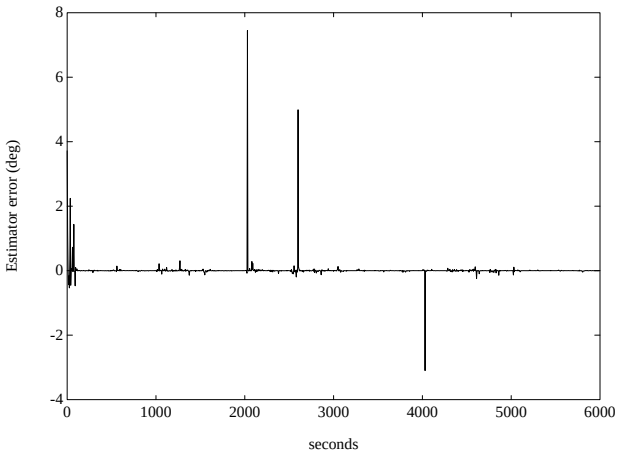

Figure 29: GAE: Estimator error $\bar{y}(k) - y(k)$

1. The size of the population used in the various genetic adaptive techniques could be allowed to vary (i.e., evolve). This could provide certain advantages in ensuring population diversity for highly nonlinear and time varying plants but the size of the population would have to remain bounded for practical implementation reasons.

2. The P model in GMBC and GMRAC could be used to predict more than one step into the future. This technique would of course have a greater dependence on an accurate P model. Furthermore, the presence of disturbances could make prediction too far into the future quite inaccurate.

3. Different underlying controllers could be tuned. Due to past experience with the cargo ship steering problem we use a PD controller exclusively and despite its simple structure it was sufficient to provide a worthwhile investigation. The GA could easily be modified to operate over more complicated linear controllers or nonlinear controllers (e.g., fuzzy controllers, conventional adaptive, or variable structure controllers). One simply has to provide a parameterization of the controller structure and represent this in the strings of the population.

4. The reference model M could be evolved by a GA so that the system is "performance adaptive" and seeks to achieve the best performance. An example of a system where such performance adaptive behavior is desirable is a reconfigurable flight control system where failures occur and the control system seeks to maintain the best, but reasonable performance.

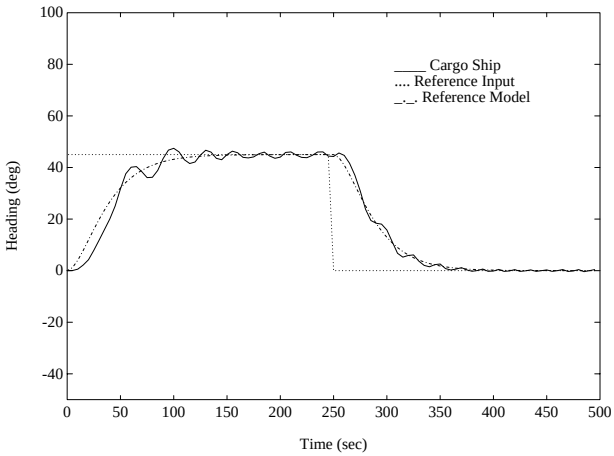

Figure 30: GAE: Cargo Ship, Close-up

5. More general population splitting schemes could be used. For instance, instead of using a fixed number of sub-populations, genetic programming [21] could be used to automatically synthesize sub-populations by treating them as tasks. Thus we could have a set of PD controllers for one operating condition of the plant and perhaps a set of dynamic controllers for another region. Alternatively, for a fixed number of sub-populations, fuzzy logic could be applied to allow for exchange of genetic material between two sub-populations.

The genetic adaptive techniques presented also raise additional issues and concerns. It cannot be overemphasized that GMBC and GMRAC are stochastic controllers. They are not guaranteed to perform in the same manner given the same reference inputs and plant initial conditions. The plots shown throughout this paper represent the response of the adaptive schemes for one trial. While all trials showed similar responses, the graphs presented here are not meant to convey any type of average behavior or guarantee of such. While the simulations in this paper show acceptable behavior, clearly there is a need for control theoretic analysis of stability, convergence and robustness properties of GMBC and GMRAC. With regard to estimator performance, it would be profitable to understand how the GA's randomness relates to "persistency of excitation" [18, 19] while the system is under closed-loop control. As described previously, it is difficult to tune the GAE system to obtain performance which is as good as that of GASC or GMRAC. Perhaps the combined GASC/GMRAC/GAE system actually needs another level of supervisory control in order to obtain better performance. Also, as this strategy is a method of indirect adaptive control, in

that we are identifying the plant parameters and the controller is based upon an estimate of the plant model, it would be useful to understand the relationships between the GASC/GMRAC/GAE scheme and conventional stochastic indirect adaptive control [18]. In addition, there is also a need to evaluate more fully the issues of computational complexity. A cursory study of the execution time required for the controller to compute the command input seemed to provide acceptable results, but no investigations were made on GASC or GAE which add complexity to GMRAC. Finally, a full evaluation of the approaches for other control applications and in a experimental test bed would be quite valuable.

References

[1] J. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michican Press, 1975.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley, 1989.

[3] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag, 1992.

[4] M. A. Lee and H. Takagi, "Integrating design stages of fuzzy systems using genetic algorithms," in *Second IEEE International Conference on Fuzzy Systems*, (San Francisco, CA.), pp. 612–617, 1993.

[5] B. Porter and M. Borairi, "Genetic design of linear multivariable feedback control systems using eigenstructure assignment," *International Journal of Systems Science*, vol. 23, no. 8, pp. 1387–1390, 1992.

[6] Z. Michelewicz, et. al, "Genetic algorithms and optimal control problems," in *Proceedings of the 29th Conference on Decision and Control*, (Honolulu, Hawaii), pp. 1664–1666, 1990.

[7] H. Ishibuchi, K. Nozaki, and N. Yamamoto, "Selecting fuzzy rules by genetic algorithm for classification problems," in *Second IEEE International Conference on Fuzzy Systems*, (San Francisco, CA.), pp. 1119–1124, 1993.

[8] Osamu Katai, et. al, "Constraint-oriented fuzzy control schemes for cart-pole systems by goal decoupling and genetic algorithms," in *Fuzzy Control Systems* (A.Kandel and G. Langholz, eds.), pp. 181–195, Boca Raton: CRC Press, 1994.

[9] C. Karr and E. Gentry, "Fuzzy control of ph using genetic algorithms," *IEEE Transactions on Fuzzy Systems*, vol. 1, no. 1, pp. 46–53, 1993.

[10] Daihee Park and Abraham Kandel and Gideon Langholz, "Genetic-based new fuzzy reasoning models with application to fuzzy control," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, no. 1, pp. 39–47, 1994.

[11] Alen Varšek and Tanja Urbančič and Bodgan Filipič, "Genetic algorithms in controller design and tuning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 5, pp. 1330–1339, 1993.

[12] H. Nomura, I. Hayashi, and N. Wakami, "A self-tuning method of fuzzy reasoning by genetic algorithm," in *Fuzzy Control Systems* (A.Kandel and G. Langholz, eds.), pp. 338–354, Boca Raton: CRC Press, 1994.

[13] R. Das and D. Goldberg, "Discrete-time parameter estimation with genetic algorithms," in *Proceedings of the 19th annual Pittsburgh Conference on Modeling and Simulation*, (Pittsburgh, PA.), pp. 2391–2395, 1988.

[14] D. Maclay and R. Dorey, "Applying genetic search techniques to drivetrain modeling," *IEEE Control Systems*, vol. 13, no. 2, pp. 50–55, 1993.

[15] K. Kristinsson and G. Dumont, "System identification and control using genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 5, pp. 1033–1046, 1992.

[16] D. Etter, M. Hicks, and K. Cho, "Recursive adaptive filter design using adaptive genetic algorithm," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, vol. 2, pp. 635–638, 1982.

[17] L. Yao and W. A. Sethares, "Nonlinear parameter estimation via the genetic algorithm," *IEEE Transactions on Signal Processing*, vol. 42, no. 4, pp. 927–935, 1994.

[18] Karl Åström and Björn Wittenmark, *Adaptive Control*, p. 356. New York: Addison-Wesley, 1989.

[19] S. Sastry and M. Bodson, *Adaptive Control: Stability, Robustness, and Convergence*. New Jersey: Prentice Hall, 1989.

[20] L. L. Porter II and K. M. Passino, "Genetic model reference adaptive control," in *Proc. IEEE International Symposium on Intelligent Control*, (Columbus, OH.), pp. 219–224, August 16-18, 1994.

[21] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge: MIT Press, 1993.

[22] J. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge: MIT Press, 1994.

[23] J. R. Layne and K. M. Passino, "Fuzzy model reference learning control for cargo ship steering," *IEEE Control Systems*, vol. 13, no. 6, pp. 23–34, Dec. 1993.

[24] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE Computer*, pp. 17–26, 1994.

La Moyne L. Porter II received a B.S. in Chemical Engineering and Electrical Engineering at Stanford University in 1991 and a M.S. in Electrical Engineering at The Ohio State University in 1994. He has worked at British Petroleum Research configuring a graphical interface for control of a chemical reactor and at Intel doing research in a manufacturing environment. He currently works at SC Solutions, an engineering consulting firm, providing control solutions for aerospace and semiconductor manufacturing firms. His interests include nonlinear systems, intelligent control, English literature, and real-time control.

Kevin M. Passino received his Ph.D. in Electrical Engineering from the University of Notre Dame in 1989. He has worked on control systems research at Magnavox Electronic Systems Co. and McDonnell Aircraft Co. He spent a year at Notre Dame as a Visiting Assistant Professor and is currently an Associate Professor in the Dept. of Electrical Engineering at The Ohio State University. He has served as a member of the IEEE Control Systems Society Board of Governors; has been an Associate Editor for the IEEE Transactions on Automatic Control; served as the Guest Editor for the 1993 IEEE Control Systems Magazine Special Issue on Intelligent Control; and a Guest Editor for a special track of papers on Intelligent Control for IEEE Expert Magazine in 1996; and was on the Editorial Board of the Int. Journal for Engineering Applications of Artificial Intelligence. He is currently the Chair for the IEEE CSS Technical Committee on Intelligent Control and is an Associate Editor for IEEE Trans. on Fuzzy Systems. He was a Program Chairman for the 8th IEEE Int. Symp. on Intelligent Control, 1993 and was the General Chair for the 11th IEEE Int. Symp. on Intelligent Control. He is co-editor (with P.J. Antsaklis) of the book "An Introduction to Intelligent and Autonomous Control," Kluwer Academic Press, 1993; co-author (with S. Yurkovich) of the book "Fuzzy Control," Addison Wesley Longman Pub., 1998; and co-author (with K.L. Burgess) of the book "Stability Analysis of Discrete Event Systems," John Wiley and Sons, 1998. His research interests include intelligent systems and control, adaptive systems, stability analysis, and fault tolerant control.