# Intelligent Control for Brake Systems

William K. Lennon and Kevin M. Passino, *Senior Member, IEEE*

***Abstract*—There exist several problems in the control of brake systems including the development of control logic for antilock braking systems (ABS) and "base-braking." Here, we study the base-braking control problem where we seek to develop a controller that can ensure that the braking torque commanded by the driver will be achieved. In particular, we develop a "fuzzy model reference learning controller," a "genetic model reference adaptive controller," and a "general genetic adaptive controller," and investigate their ability to reduce the effects of variations in the process due to temperature. The results are compared to those found in previous research.**

***Index Terms*—Adaptive control, automotive, brakes, fuzzy control, genetic algorithms.**

## I. Introduction

AUTOMOTIVE antilock braking systems (ABS) are designed to stop vehicles as safely and quickly as possible. Safety is achieved by maintaining lateral stability (and hence steering effectiveness) and trying to reduce braking distances over the case where the brakes are controlled by the driver. Current ABS designs typically use wheel speed compared to the velocity of the vehicle to measure when wheels lock (i.e., when there is "slip" between the tire and the road) and use this information to adjust the duration of brake signal pulses (i.e., to "pump" the brakes). Essentially, as the wheel slip increases past a critical point where it is possible that lateral stability (and hence our ability to steer the vehicle) could be lost, the controller releases the brakes. Then, once wheel slip has decreased to a point where lateral stability is increased and braking effectiveness is decreased, the brakes are reapplied. In this way the ABS cycles the brakes to try to achieve an optimum tradeoff between braking effectiveness and lateral stability. Inherent process nonlinearities, limitations on our abilities to sense certain variables, and uncertainties associated with process and environment (e.g., road conditions changing from wet asphalt to ice) make the ABS control problem challenging. Many successful proprietary algorithms exist for the control logic for ABS. In addition, several conventional nonlinear control approaches have been reported in the open literature (see, e.g., [1] and [2]), and even one intelligent control approach has been investigated [3].

Manuscript received July 29, 1996; revised September 29, 1997. Recommended by Associate Editor, K. Passion. This work was supported in part by Delphi Chassis Division of General Motors, the Center for Automotive Research (CAR) at Ohio State University, and National Science Foundation under Grants IRI9210332 and EEC9315257.

The authors are with the Department of Electrical Engineering, Ohio State University, Columbus, OH 43210 USA.

Publisher Item Identifier S 1063-6536(99)01618-8.

In this paper, we do not consider brake control for a "panic stop," and hence for our study the brakes are in a non-ABS mode. Instead, we consider what is referred to as the "base-braking" control problem where we seek to have the brakes perform consistently as the driver (or an ABS) commands, even though there may be aging of components or environmental effects (e.g., temperature or humidity changes) which can cause "brake grab" or "brake fade." We seek to design a controller that will try to ensure that the braking torque commanded by the driver (related to how hard we hit the brakes) is achieved by the brake system. Clearly, solving the base braking problem is of significant importance since there is a direct correlation between safety and the reliability of the brakes in providing the commanded stopping force. Moreover, base braking algorithms would run in parallel with ABS controllers so that they could also enhance braking effectiveness while in ABS mode.

Prior research on the braking system considered here has shown that one of the primary difficulties with the brake system lies in compensating for the effects of changes in the "specific torque," to be defined below, that occur due to temperature variations in the brake pads [4]. Previous research on this system has been conducted using proportional-integral-derivative (PID), lead-lag, autotuning, and model reference adaptive control (MRAC) techniques [5]. While several of these techniques have been highly successful (particularly the lead-lag compensator that we use as a base-line comparison here), there is still a need to improve the compensators for the case where there are changes in specific torque due to temperature variations that result from, for example, repeated application of the brakes. In this paper we investigate the performance of three intelligent control techniques, fuzzy model reference learning control [6], genetic model reference adaptive control [7], [8], and "general genetic adaptive control" [9], for the base braking problem and compare their performance to the best results found in [4] and [5]. We especially focus on the performance of these techniques when there are variations in the specific torque.

In Section II we provide a simulation model for the base braking system that has proven to be very effective in developing, *implementing*, and testing control algorithms for the actual braking system [4], [5]. In Section III we develop a fuzzy model reference learning controller (FMRLC) for the base braking system problem. Its performance is evaluated in simulation by comparing it to a lead-lag compensator from [5] under varying specific torque conditions. In Sections IV and V, we develop a genetic model reference adaptive controller (GMRAC) and a general genetic adaptive controller (GGAC)

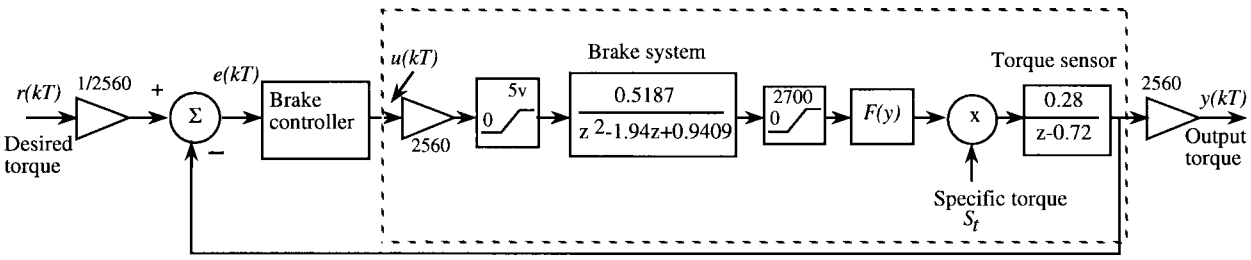



Fig. 1. Base braking control system.

for the base braking problem. We use similar test conditions for evaluating the controller and compare its performance to the previous ones. Numerical performance results are shown in Section VI, and some concluding remarks are provided in Section VII.

## II. The Base Braking Control Problem

Fig. 1 shows the diagram of the base braking system, as developed in [5]. The input to the system, denoted by $r(kT)$, is the braking torque requested by the driver. The output, $y(kT)$ (in ft-lbs), is the output of a torque sensor, which directly measures the torque applied to the brakes. Note that while torque sensors are not available on current production vehicles, there is significant interest in determining the advantages of using such a sensor. The signal $e(kT)$ represents the error between the reference input and output torques, which is used by the controller to create the input to the brake system, $u(kT)$. A sampling interval of $T = 0.005$ s was used for all our investigations.

The General Motors braking system used in this research is physically limited to processing signals between [0, +5] V, while the braking torque can range from 0 to 2700 ft·lb. For this reason and other hardware specific reasons [5], the input torque is attenuated by a factor of 2560 and the output is amplified by the same factor. After $u(KT)$ is multiplied by 2560 it is passed through a saturation nonlinearity where if $2560u(kT) \leq 0$, the brake system receives a zero input and if $2560u(kT) \geq 5$ then the input is five. The output of the brake system passes through a similar nonlinearity that saturates at zero and 2700. The output of this nonlinearity passes through $F(y)$, which is defined as

$$F(y) = \frac{y}{2502.4419} + 0.0139878.$$

The function $F(y)$ was experimentally determined and represents the relationship between brake fluid pressure and the stopping force on the car. Next, $F(y)$ is multiplied by the specific torque $S_t$. This signal is passed through an experimentally determined model of the torque sensor; the signal is scaled and $y(kT)$ is output. The specific torque $(S_t)$ in the braking process reflects the variations in the stopping force of the brakes as the brake pads increase in temperature. The stopping force applied to the wheels is a function of the pressure applied to the brake pads and the coefficient of friction between the brake pads and the wheel rotors. As the brake pads and rotors increase in temperature, the coefficient of friction between the brake pads and the rotors increases. As a result, less pressure on the brake pads is required for the same amount of braking force. The specific torque $S_t$ of this braking system has been found experimentally to lie between two limiting values so that

$$0.85 \leq S_t \leq 1.70.$$

All the simulations we conducted use a repeating 4 s input reference signal. The input reference begins at 0 ft·lb, increases linearly to 1000 ft·lb by 2 s, and remains constant at 1000 ft·lb until 4 s. After 4 s the states of the brake system are cleared (i.e., set to zero), and the simulation is run again. The first two 4-s simulations are run with $S_t = 0.85$, corresponding to "cold brakes" (a temperature of 125° F). The next two 4-s simulations are run with $S_t$ increasing linearly from 0.85 at 0 s to 1.70 by 8 s. Finally, two more 4-s simulations are run with $S_t = 1.7$, corresponding to "hot brakes" (a temperature of 250°F). Fig. 2 shows the reference input and the specific torque over the course of the simulation that we will use throughout this paper.

As a base-line comparison for the techniques to be shown here, the results of a conventional lead-lag controller are shown in Fig. 3. This controller was chosen because it was the best conventional controller previous researchers [4], [5] developed for this braking simulation. The lead-lag controller is defined as

$$\frac{u(kT)}{e(kT)} = \frac{0.2(z - 0.95 + 0.02j)(z - 0.95 - 0.02j)}{z(z-1)}.$$

As can be seen in Fig. 3, the conventional controller performs adequately at first, but as the specific torque increases, the controller induces a large overshoot at the beginning of each ramp-step in the simulation. The conventional controller does not compensate for the increased specific torque of the brakes and hence overshoots the reference input when the reference input is small. In fairness to the conventional method however, it was designed for cold brakes. If it were designed for hot brakes it would perform better for $S_t = 1.7$, but then it would not perform as well for cold brakes. Clearly, there is a need for an adaptive or robust controller.

Past researchers have investigated certain adaptive methods. First a gradient-based MRAC was studied [5]. This controller performed worse than the controllers shown in this paper (it had a much poorer transient response) so we still use the fixed lead-lag compensator for comparisons. In [4] and [5] the

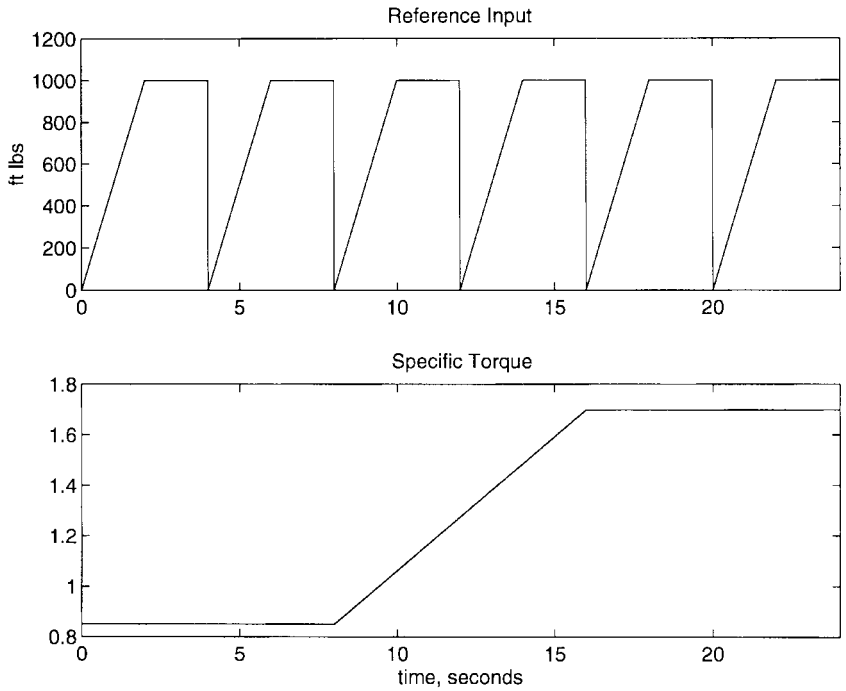



Fig. 2. Reference input and specific torque.

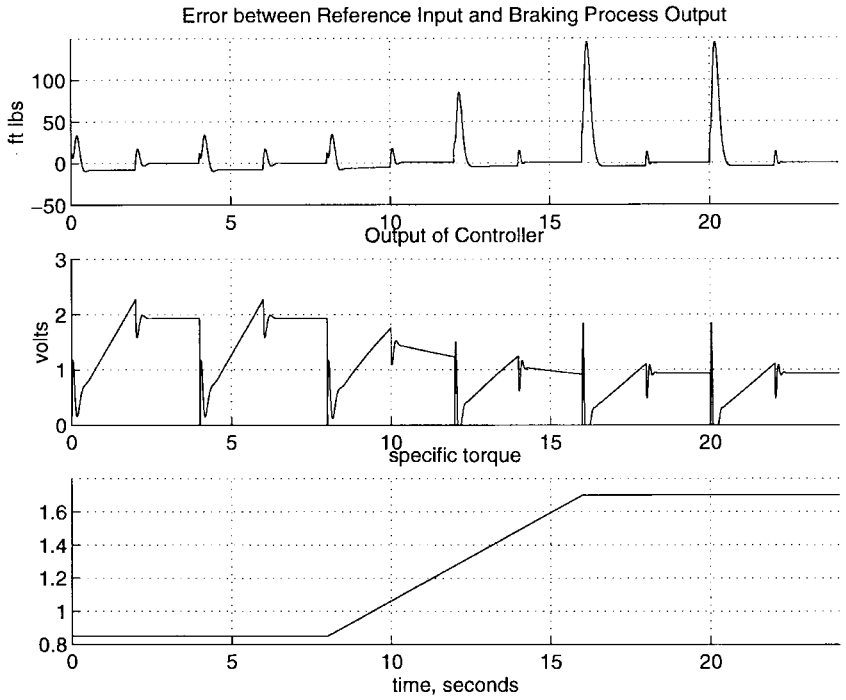



Fig. 3. Results using conventional lead-lag controller.

authors also investigated the use of a proportional-integral-derivative (PID) autotuner. This method was very successful in the off-line tuning (i.e., open-loop tuning) of a PID brake controller. Its only drawback is that it does not provide for continuous adaptation as the temperature changes (which is our primary concern here).

The intelligent control techniques to be presented in this paper use a reference model to quantify the desired performance of the closed-loop system. This model was developed in previous work [5], and is defined as shown in (1)

$$\frac{y_m(kT)}{r(kT)} = \frac{0.1(z-0.95)}{(z-0.7005)(z-0.8808+0.498j)(z-0.8808-0.498j)}. \quad (1)$$

This model was chosen to represent reasonable and adequate performance objectives for the brake system. The physical

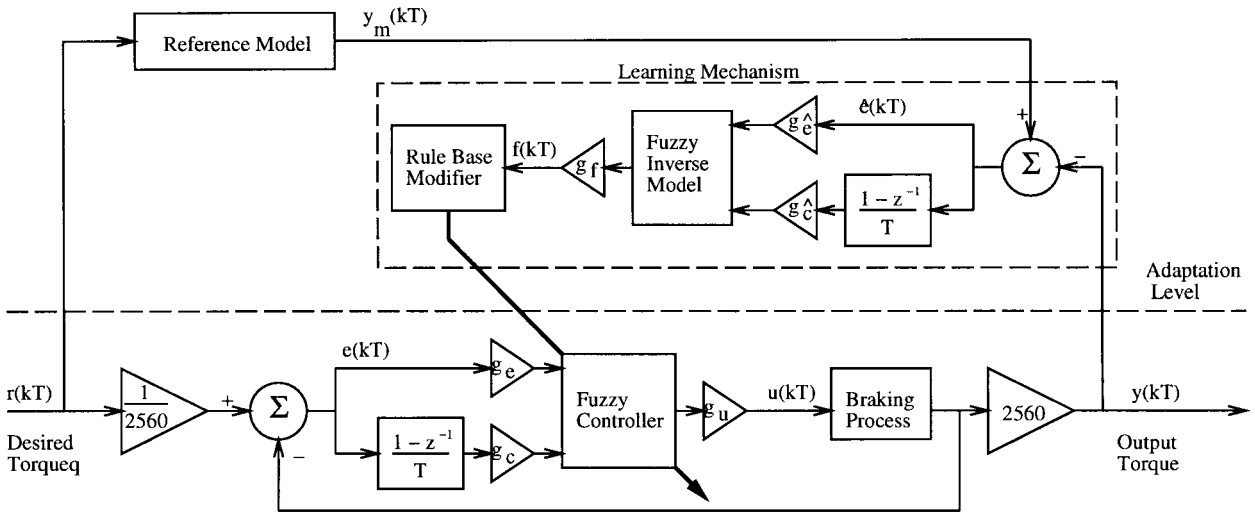



Fig. 4. FMRLC for base braking.

process was taken into consideration in the development of this model.

## III. Adaptive Fuzzy Control

In this section we develop an adaptive fuzzy controller for the base braking problem. In particular, we modify the fuzzy model reference learning control (FMRLC) technique [3], [6], [10], [11] that has already been utilized in several applications. The FMRLC, shown in Fig. 4, utilizes a learning mechanism that observes the behavior of the fuzzy control system, compares the system performance with a model of the desired system behavior, and modifies the fuzzy controller to more closely match the desired system behavior. Next we describe each component of the FMRLC in more detail.

### A. FMRLC for Base Braking

The inputs to the fuzzy controller are the error $e(kT)$ and change in error $c(kT)$ defined as $e(kT) = r(kT) - y(kT)$ and $c(kT) = (e(kT) - e(kT-T))/T$, respectively. The gains $g_e$, $g_c$, and $g_u$ were adjusted to normalize the universe of discourse (the range of values for an input or output variable) so that all possible values of the variables lie between $[-1, 1]$. After some simulation-based investigations we chose $g_e = 25$, $g_c = 3$, and $g_u = 0.002$.

The knowledge-base for the fuzzy controller is generated from IF–THEN control rules. A set of such rules forms the "rule base" which characterizes how to control a dynamical system. The fuzzy controller we designed consists of two inputs with six membership functions each, as shown in Fig. 5. Thus, there are a total of 36 IF–THEN rules in the fuzzy controller of Fig. 4. There are 36 triangular output membership functions that peak at one, are symmetric, and have base widths of 0.2. It is the centers of the output membership functions that are adjusted by the FMRLC.

Suppose that, as shown in Fig. 5, the normalized inputs to the fuzzy controller are $\overline{e}(kT) = g_e e(kT) = 0.293$ and $\overline{c}(kT) = g_c c(kT) = -0.546$. Let us use the linguistic description "error" for $\overline{e}(kT)$, "change in error" for $\overline{c}(kT)$, and "output" for $\overline{u}(kT)$. Therefore, in the example shown in Fig. 5, the certainty that "error is positive small" is 0.764, and the certainty that "error is positive medium" is 0.236. Likewise, the statement "change in error is negative medium" has a certainty of 0.873 and "change in error is negative small" has a certainty of 0.127. All other values have certainties of zero. Of the 36 rules in the fuzzy controller rule-base, only four have premises with certainties greater than zero (we use the min operator to represent the "and" operator in the premise). They are as follows.

**If** *error* is *positive-small* **and** *change-in-error* is *negative-small* **Then** *output* is *consequence*$^{4,3}$
**If** *error* is *positive-small* **and** *change-in-error* is *negative-medium* **Then** *output* is *consequence*$^{4,2}$
**If** *error* is *positive-medium* **and** *change-in-error* is *negative-small* **Then** *output* is *consequence*$^{5,3}$
**If** *error* is *positive-medium* **and** *change-in-error* is *negative-medium* **Then** *output* is *consequence*$^{5,2}$.

Here *consequence*$^{i,j}$ is the linguistic value associated with the output membership function of the rule. The membership function of the implied fuzzy set corresponding to each *consequence*$^{i,j}$ is determined by taking the minimum of the certainty of the premise with the membership function associated with *consequence*$^{i,j}$. The implied fuzzy sets are shown in the shaded regions in Fig. 5.

The output of the fuzzy controller, $u(kT)$, is computed via the center of gravity (COG) defuzzification algorithm, as shown in Fig. 5. For COG the certainty of each rule premise is calculated, and the trapezoid with a height equal to that certainty is shaded. The center of gravity of the shaded regions is calculated, and that is the output of the fuzzy controller, $\overline{u}(kT)$.

In the FMRLC, typically the center of the output membership function of each rule is initialized to zero when the simulation is first started. This is done to signify that the direct fuzzy controller initially has "no knowledge" of how to control the brake system (of course it does have some knowledge since the designer must specify everything else in the fuzzy

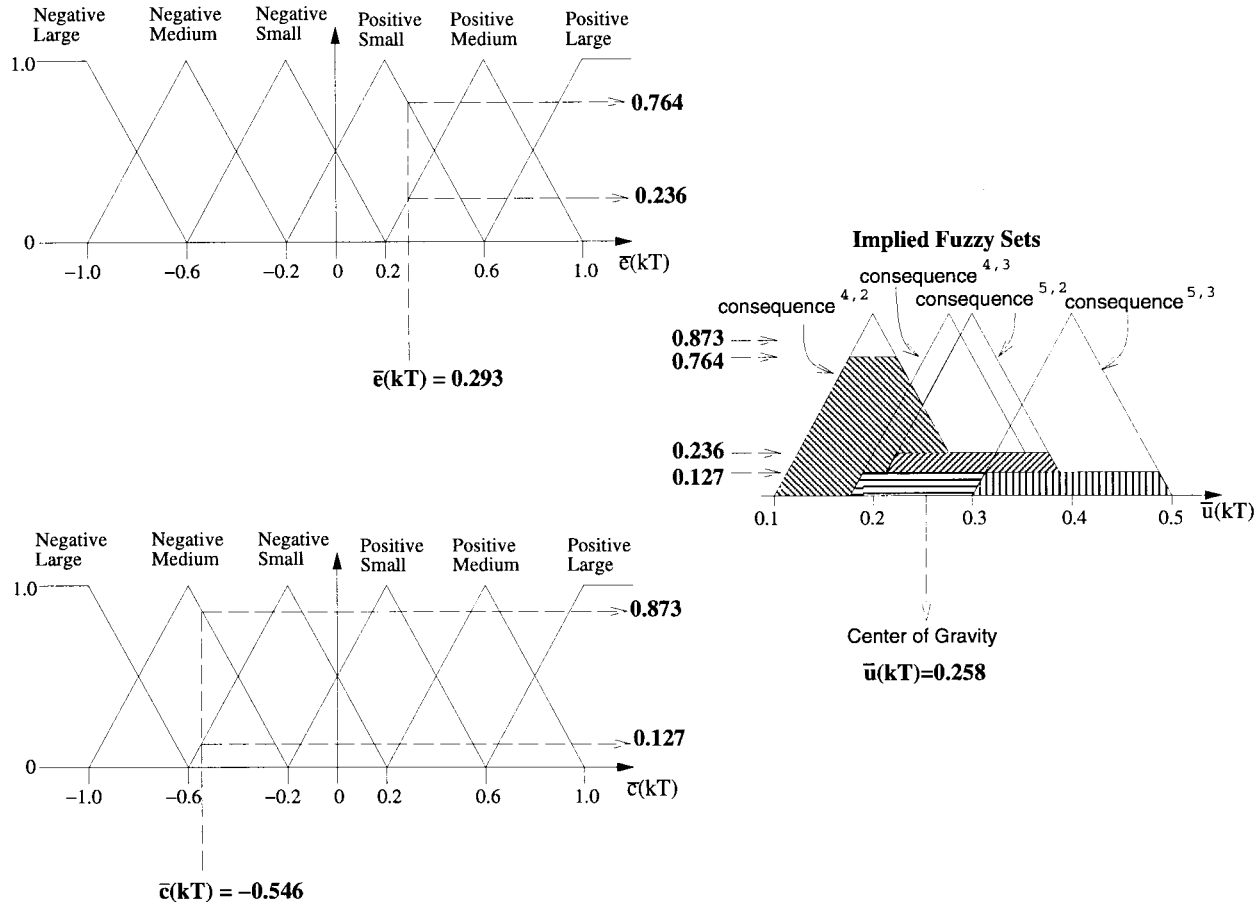



Fig. 5. Example of fuzzy inference.

controller except for the output membership function centers). Note that using a well-tuned direct fuzzy controller for the braking system to initialize the FMRLC only slightly affects performance. Of course, we are not implying that we are not using model information about the plant to perform our overall design of the adaptive fuzzy controller. Plant information is used in the tuning of the FMRLC through an iterative process of simulation of the closed-loop system and tuning of the scaling gains. More details are provided on this below.

The output of the fuzzy controller can be mapped as a three-dimensional surface, where the two inputs, $\overline{e}(kT)$ and $\overline{c}(kT)$, define the $X$ and $Y$ axes and the output of the fuzzy controller, $\overline{u}(kT)$ defines the $Z$ axis. The rule base of the fuzzy controller can be visualized in terms of this surface, and the learning mechanism of the FMRLC can be thought of as adjusting the contours of this surface.

The rules in the "fuzzy inverse model" quantify the inverse dynamics of the process [3], [6], [10], [11]. The fuzzy inverse model is very similar to the direct fuzzy controller in that it has two inputs, error and change in error, one output, and a knowledge base of IF–THEN rules (their membership functions have shapes similar to those shown in Fig. 5). Both inputs in our fuzzy inverse model contain membership functions (with the middle one centered at zero), corresponding to 25 IF–THEN rules. The fuzzy inverse model operates on the error, $\hat{e}(kT)$, and change in error, $\hat{c}(kT)$, between the desired behavior of the system, $y_m(kT)$, and the observed behavior of the system, $y(kT)$. It then computes what the input to the braking process *should have been* to drive this error to zero. This information is passed to the rule-base modifier which then adjusts the fuzzy controller to reflect this new knowledge.

The output centers of the rule base of the fuzzy inverse model are structured so that small differences between the desired and observed system behavior $(g_{\hat{e}}\hat{e}(kT) \approx 0)$ result in fine tuning of the fuzzy controller rule-base, while large differences $(g_{\hat{e}}\hat{e}(kT) \approx 1)$ result in very large adjustments to the fuzzy controller rule-base. For example, the following are three of the 25 rules in the fuzzy inverse model.

**If** *error* is *zero* **and** *change-in-error* is *zero* **Then** *output* is *zero*
**If** *error* is *positive-small* **and** *change-in-error* is *positive-small* **Then** *output* is *positive-tiny*
**If** *error* is *positive-large* **and** *change-in-error* is *positive-large* **Then** *output* is *positive-huge*

Here the input linguistic values *zero*, *positive-small*, and *positive-large* correspond to numerical values of 0, 0.5, and 1.0, respectively, and the output linguistic vales *zero*, *positive-tiny*, and *positive-huge* correspond to numerical values of 0, 0.031, and 1.0. After some simulation-based investigations we chose $g_{\hat{e}} = 0.5$, $g_{\hat{c}} = 0.015$, and $g_f = 0.0005$.

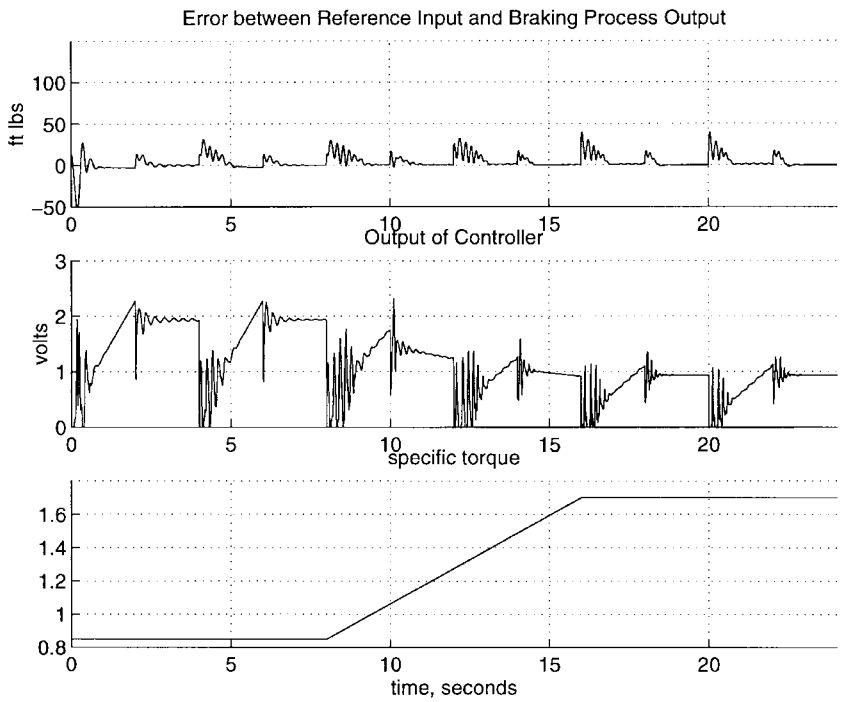



Fig. 6. Results using the FMRLC.

The rule base modifier uses the information from the fuzzy inverse model $f(kT)$, to change the rule base of the direct fuzzy controller. At each time, the direct fuzzy controller computes the implied fuzzy set of each of the 36 rules in its knowledge base. Because there are two inputs, at any one time sample at most four rules will be activated. The rule-base modifier stores the degree of certainty of each rule premise for the past several time samples. It then modifies those rules by a factor equal to the output $f(kT)$ times the degree of certainty of the rule (note that this approach to knowledge-base modification differs slightly from that in [6] since there the premise certainties are not used). Because there is a delay of three time samples in the dynamics of the braking process, the rule base modifier adjusts the centers of the rules that were active three time units in the past. In this manner, the learning mechanism adjusts the rules that caused the error in the braking process, and not just the rules that were active most recently. For more details on the operation and design of the FMRLC see [11].

### *B. FMRLC Results*

The results of the FMRLC simulation are shown in Fig. 6. The FMRLC does not perform well initially because it is learning to control the braking process. The FMRLC was more successful in the remaining seconds and outperformed the conventional controller when the specific torque of the brake system increased. Note in Fig. 6 that the FMRLC very quickly (within 0.25 s) learned to control the braking process. The FMRLC performed consistently well over the full range of specific torque. It is difficult to discern any difference between cold and hot brakes in the output of the braking process (which is the goal of this project). Nevertheless, it is clearly seen (by comparing the controller output in the first 8 s to the controller output in the final 8 s) that the fuzzy controller is learning to compensate for the increased specific torque by sending a smaller signal to the brakes. We computed the error between the reference input and the braking process output at each time step in the simulation. This error was squared and summed over the entire simulation. These results are shown in Table I in Section VI.

Fig. 7 shows the nonlinear map implemented by of the fuzzy controller in Fig. 4 at four time instances. The black "X" was included on the 3-dimensional plots to clearly show the center of the fuzzy surface, where most of the learning takes place. When the FMRLC is first run, it quickly creates inference rules that effectively control the braking process. As the simulation continues and the dynamics of the plant change, (i.e., the specific torque increases), the FMRLC tunes the rules of the fuzzy controller to adequately compensate for the change in brake dynamics. Note that at first glance of Fig. 7, the surface of the fuzzy controller does not appear to change appreciably as the braking process changes. This is because much of what the controller has learned is still valid and the learning mechanism does not affect these areas. However, it is important to see that the center of the fuzzy surface, corresponding to $e(kT) \approx 0$ and $c(kT) \approx 0$, (the area in which the controller is designed to operate), decreases by roughly one-half when the specific torque of the braking process increases by two. Thus the FMRLC learns to adapt to conditions of the braking process, keeping old rules unmodified and adjusting only those rules that are used in the present operating conditions. In the simulation, as the brake pads increase in temperature and the specific torque of the brakes increase, the learning mechanism adjusts the rules of the fuzzy controller to compensate for the increased gain in the braking process.

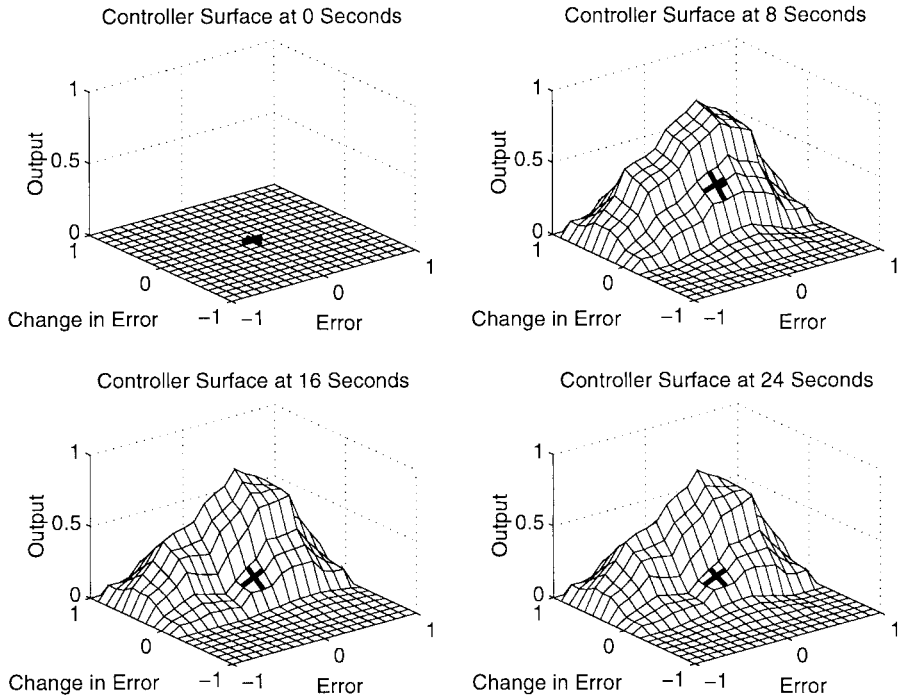



Fig. 7. Adaptation of fuzzy controller surface. The axes labeled "error" is $\overline{e}(kT)$, "change in error" is $\overline{c}(kT)$, and "output" is $\overline{u}(kT)$. Note that after the controller surface is first created, the only significant modifications occur at the center of the surface, marked by the "X." The surface center adapts as the brake process changes, from a height of 0.378 when $S_t = 0.85$ to a height of 0.183 when $S_t = 1.70$.

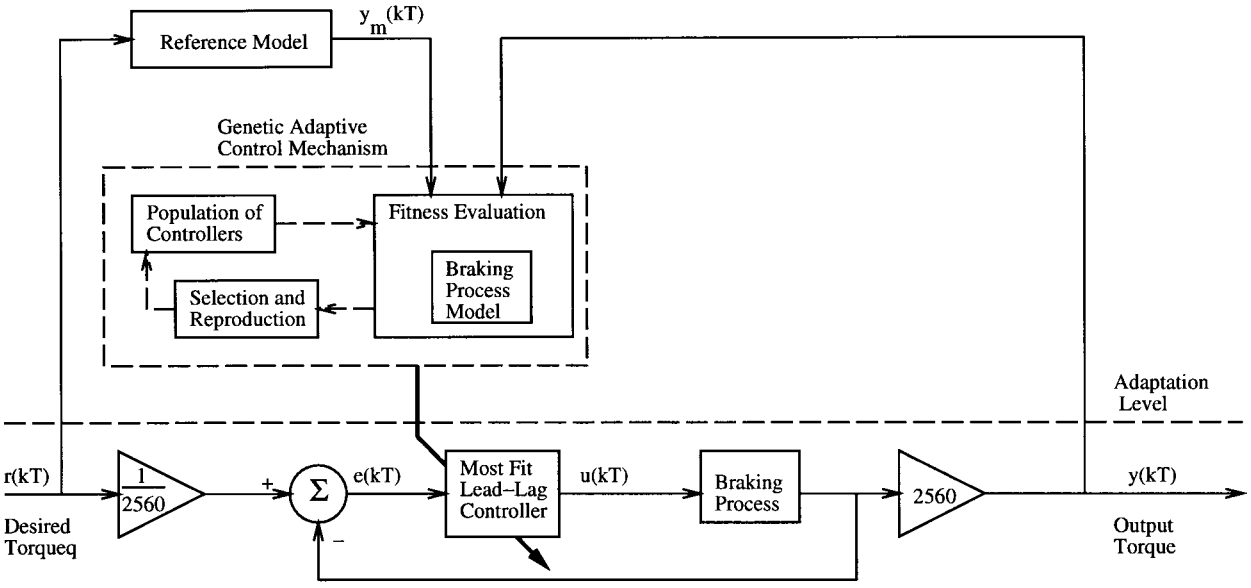



Fig. 8. GMRAC for base braking.

## IV. Direct Genetic Adaptive Control

In this section we develop a genetic model reference adaptive controller (GMRAC) [7], [8] for the base braking control problem. A genetic algorithm (GA) is used to evolve a good brake controller as the operating conditions of the braking process change. The GMRAC, shown in Fig. 8, uses a simplified model of the braking process to evaluate a population (set) of braking controllers and "evolve" a good controller for the braking process. Next we describe each component of the GMRAC in more detail, but first we briefly outline the basic mechanics of GA's.

### A. The Genetic Algorithm

A genetic algorithm is a parallel search method that manipulates a string of numbers (a "chromosome") according to the laws of evolution and biology. A population of chromosomes are "evolved" by evaluating the fitness of each chromosome and selecting members to "reproduce" based on their fitness.

Evolution of the population of individual chromosomes here is based on four genetic operators: crossover, mutation, selection, and elitism.

Selection is the process where the most fit individuals survive to reproduce and the weak individuals die out. The selection process evaluates each chromosome by some fitness mechanism and assigns it a fitness value. Those individuals deemed "most fit" are then selected to become parents and reproduce. The selection of which chromosomes will reproduce is not deterministic, however. Every member of the population has a probability of being selected for reproduction equal to its fitness divided by the sum of the fitness of the population. Hence, the more fit individuals have a greater change to reproduce than the less fit individuals. Crossover is the procedure where two "parent" chromosomes exchange genetic information (i.e., a section of the string of numbers) to form two chromosome offspring. Crossover can be considered a form of local search in the population space. Mutation is a form of global search where the genetic information of a chromosome is randomly altered. Elitism is used in the GMRAC to ensure that the most fit member of the population is moved without modification into the next generation. By including elitism, we can increase the rates of crossover and mutation, thereby increasing the breadth of search, but still ensure that a good controller remains present in the population.

Our genetic algorithm uses the base-10 number system as opposed to base-2 which is commonly used in [12] and [13]. While base-2 systems can be advantageous because they consist of smaller "genetic building blocks," they have the disadvantage of more complicated encoding/decoding procedures and longer strings (which can affect our ability to implement the genetic adaptive controllers in real time). While both bases work well, we chose to use base-10 because of the ease in which controller parameters can be coded into a chromosome, as described below.

### B. GMRAC for Base Braking

In this section, we describe each component of the genetic adaptive mechanism in Fig. 8.

*1) The Population of Controllers:* The GMRAC uses a lead-lag controller which is the best conventional controller previous researchers in [4] and [5] have found for this braking simulation. The transfer function of this controller is

$$\frac{u(kT)}{e(kT)} = \frac{K(z-0.95+0.02j)(z-0.95-0.02j)}{z(z-1)}.$$

The gain of the controller was constant at $K = 0.2$ in previous research, but will be "evolved" by the GMRAC to adapt to braking process changes. The range of valid gains has been limited to $(0.0 \leq K \leq 0.4)$. This is to try to ensure that the GA does not evolve controllers that are unstable $(K < 0)$ or highly oscillatory $(K > 0.4)$.

The controller population size was constant at eight members. This was a compromise between search speed and processing time. In general, as the population size increases, more variety exists in the population and therefore "good" controllers are more likely to be found. However, computation time is greatly affected by population size, and therefore the maximum population size is limited by the speed of the processor and the sampling interval of the system. Note that performance of the GMRAC was not significantly affected by population sizes of six or more. Rather, the GMRAC performance was more greatly affected by the crossover probability, mutation probability, and the number of time units into the future the fitness evaluation attempts to predict (described below).

Each individual controller gain was described by a three-digit base-10 number. Each digit is called a "gene" and the string of genes together forms the "chromosome." This chromosome is very simply decoded into a decimal number corresponding the gain $K$ of the lead-lag controller. To decode a chromosome, simply place a decimal point before the first gene of the chromosome. For example, a chromosome of [345] would decode into $K = 0.345$.

*2) Fitness Evaluation and the Braking Process Model:* The GA uses $r(kT)$, $u(kT)$, $y(kT)$, $y_m(kT)$, (and their past values), and a plant model to evaluate the fitness of the strings in the population of candidate controllers. At each time step (i.e., each "generation") the GA chooses the controller in the population with maximum fitness value to control the plant from time $k$ to time $k+1$.

The process model used in the GMRAC is a simplified model of the braking process. The model of the plant is described by the transfer function

$$\frac{\hat{y}(kT)}{\hat{u}(kT)} = \frac{0.25}{z^3 - 2.66z^2 + 2.338z - 0.677}. \tag{2}$$

Comparing this to the actual model of the brake system in Section II, we see that this model ignores significant nonlinearities and the $S_t$ "disturbance" (i.e., we treat the model in Section II as the "truth model").

The genetic algorithm seeks to maximize the fitness function

$$J = \frac{\alpha}{\overline{J}^2 + \alpha}$$

where

$$\overline{J} = e(k) + \beta\left(\frac{\hat{e}(k+N) - e(k)}{N}\right)$$

and $\hat{e}$ is the predicted error between the outputs of the plant and reference model. Here $N$ denotes the "look ahead" time window, signifying that the fitness evaluation attempts to predict the braking process for the next $N$ unit samples. Because there is significant delay between control input and braking output, a short time window would cause the current controller candidates to be evaluated mostly on the performance of past controllers, leading to inaccurate fitness evaluations. However, longer time windows cause greater deviations between the braking process model and the actual braking process, and this also leads to inaccurate fitness evaluations. We selected $N = 10$ as a good compromise to maintain the validity of the fitness evaluations.

After some simulation-based investigations, we choose $\alpha = 0.000\,001$ and $\beta = 20$. The constant $\beta$ defines the number of time samples in which the error should reach zero. For example, if $N = 10$ and $\beta = 20$, then the fitness function is

maximized when $\hat{e}(k+10) = 0.5e(k)$, which would indicate that $e(k)$ should reach zero in $\beta = 20$ time steps.

The fitness evaluation proceeds according to the following pseudocode.

1) Collect $r(k)$, $y(k)$, and $y_m(k)$.
2) Compute a first-order approximation of $y_m$, $\hat{y}_m(k+N) = y_m(k) + N[y_m(k) - y_m(k-1)]$.
3) Estimate the closed-loop system response for the next $N$ samples for each controller in the population:
   For $j = 1$ to $N$:
   Generate $\hat{y}(k+j)$ from braking process model in (2).
   Estimate $\hat{r}(k+j) = r(k) + j[r(k) - r(k-1)]$.
   Compute $\hat{e}(k+j) = \hat{r}(k+j) - \hat{y}(k+j)$.
   Generate $\hat{u}(k+j+1)$ from $\hat{e}(k+j)$, $\hat{e}(k+j-1)$, etc., and controller parameters.
   Next $j$.
4) Compute $\epsilon(k+N) = \hat{y}_m(k+N) - \hat{y}(k+N)$.
5) Assign fitness, $J_i$, to each controller candidate, $C_i$:
   Let $\overline{J}_i = e(k) + \beta((\epsilon(k+N) - e(k))/N)$
   $J_i = \alpha/(\overline{J}_i^2 + \alpha)$.
6) The maximally fit controller becomes the next controller used between times $k$ and $k+1$. The selection, crossover, mutation, and elitism [7] processes are applied to produce the next generation of controllers (see below). Increment the time index and go to Step 1).

*3) Selection and Reproduction of Controllers:* Once each controller in the population has been assigned a fitness $J_i$, the GA uses the "roullete-wheel" selection process [12] to determine which controllers will reproduce into the next generation. The roullete-wheel selection process picks the "parents" of the next generation in a manner similar to spinning a roullete-wheel, with each individual in the population assigned an area on the roullete-wheel proportional to that individual's fitness. Hence the probability that an individual will be selected as a particular parent of the next generation is proportional to the fitness of that individual. Note that some individuals will likely be selected more than once (indicating they will have more than one offspring), while other individuals will not be selected at all. In this way the "bad" controllers are generally removed from the population.

Next, the parents are coupled together and generally undergo crossover. The probability that crossover occurs between two parents is determined *a priori* by a crossover probability. In our simulation, two parents will undergo crossover with probability 0.90. Crossover is conducted differently than is commonly described. In all genetic algorithms used in these simulations, crossover is not done by selecting a crossover site and exchanging genes beginning at the crossover site and ending at the end of the chromosome. Instead, crossover is done on a gene-by-gene basis. Each gene (digit) in the chromosome has a 0.5 probability of being exchanged for the digit in the same location on the mating chromosome. For example, the GA uses a string length of three, so two possible parent chromosomes could be [333] and [111]. If these two chromosomes undergo crossover, possible offspring pairs could be [113] and [331] or [131] and [313].

After crossover, the two offspring undergo mutation, with a prespecified probability. In the GMRAC, we used a mutation probability of 0.3, which means every digit in the chromosome has a 30% probability of being mutated. Note that this is a relatively high mutation probability, but with the elitism operator ensuring that a good controller is always in the population, a high mutation rate helps to offset the small population size and improve the searching ability of the GA. Moreover, we have found that since the fitness function $J_i$ is time varying and the plant is changing in real time, there is a significant need to make the GA aggressive in exploring various regions (i.e., in trying different controller candidates). If it locks on to some controller parameter values and is inflexible to change it will not be successful at adaptation.

### C. GMRAC Results

Fig. 9 shows the results of the braking simulation using the GMRAC. As can be seen in Fig. 9, the GMRAC performs more consistently as the specific torque of the brakes increases. While the performance does degrade somewhat as specific torque increases, at its worst it is still significantly better than the conventional controller. Note that contrary to conventional controllers and the FMRLC discussed previously, the GMRAC is stochastic, and the results in Fig. 9 represent the behavior for only one simulation run. We did, however, find similar average behavior when we performed 100 simulation runs. We computed the error between the reference input and the braking process output at each time step in the simulation. This error was squared and summed over the entire simulation. The minimum, average, and maximum errors for the 100 simulations are shown in Table I in Section VI.

### D. GMRAC with Fixed Population Members

Because genetic algorithms are stochastic processes, there is always a small possibility that good controllers will not be found and hence degrade performance. While this possibility diminishes with population size and the use of elitism, it nevertheless exists. One method to combat this possibility is to seed the population of the GA with individuals that remain unchanged in every generation. These fixed controllers can be spaced throughout the control parameter space to ensure that a reasonably good controller is always present in the population. Simulations were run for the GMRAC with three fixed controllers in the GA population (leaving the remaining five controllers to be adapted by the GA as usual). Because the controller gains were restricted to $-0.4 < k \leq 0$, the population was seeded with three fixed PD controllers, defined by $k = 0$, $k = 0.2$, and $k = 0.4$. Because the fixed controllers adequately cover the parameter space, the mutation probability of the GA was be decreased to 0.1.

Using fixed controllers is a novel control technique that appears to decrease the variations in the performance results. The technique is conceptually similar to [14] where Narendra and Balakrishnan use fixed plant models in an indirect adaptive controller to identify a plant and improve transient responses. Likewise, having a genetic algorithm with fixed controller

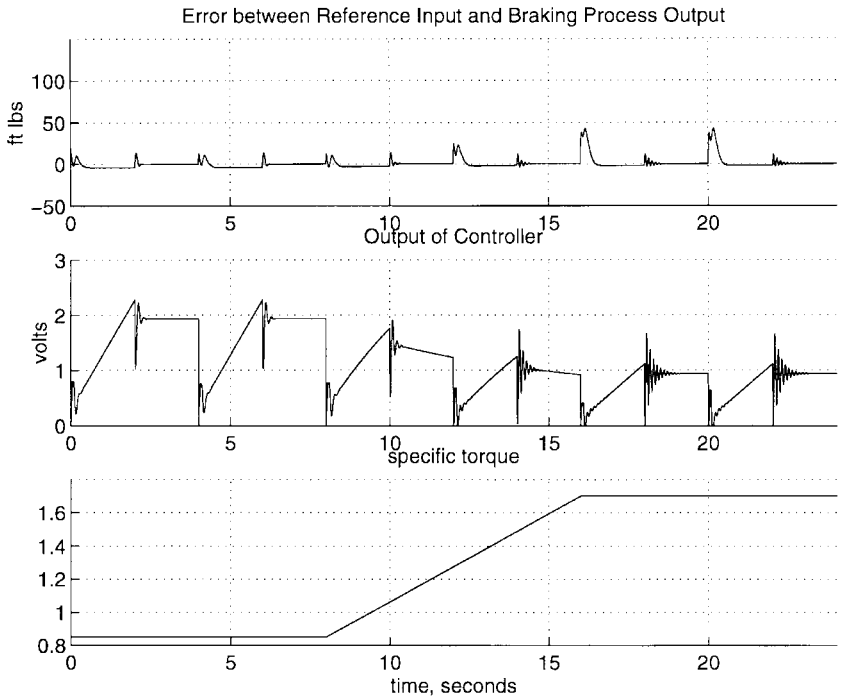



Fig. 9. Results using GMRAC.

TABLE I
RESULTS

| Control Technique | Simulation time (seconds) | | | | | |
|---|---|---|---|---|---|---|
| | 0-4 | 4-8 | 8-12 | 12-16 | 16-20 | 20-24 |
| | $S_t = 0.85$ | | $0.85 \leq S_t \leq 1.7$ | | $S_t = 1.7$ | |
| Conventional Lead-Lag | 0.00843 | 0.00843 | 0.00716 | 0.3552 | 0.11761 | 0.11761 |
| FMRLC | 0.01299 | 0.00732 | 0.00886 | 0.00968 | 0.00962 | 0.00864 |
| GMRAC | | | | | | |
| minimum | 0.00136 | 0.00137 | 0.00098 | 0.00154 | 0.00344 | 0.00356 |
| average | 0.00170 | 0.00166 | 0.00120 | 0.00390 | 0.01589 | 0.01525 |
| maximum | 0.00260 | 0.00304 | 0.00166 | 0.02137 | 0.07151 | 0.06087 |
| GMRAC with Fixed Controllers | | | | | | |
| minimum | 0.00161 | 0.00160 | 0.00118 | 0.00169 | 0.00435 | 0.00435 |
| average | 0.00161 | 0.00161 | 0.00121 | 0.00186 | 0.00446 | 0.00446 |
| maximum | 0.00166 | 0.00167 | 0.00122 | 0.00235 | 0.00574 | 0.00556 |
| GGAC with Fixed Controllers | | | | | | |
| minimum | 0.00459 | 0.00479 | 0.00401 | 0.00338 | 0.00497 | 0.00499 |
| average | 0.00601 | 0.00553 | 0.00459 | 0.00358 | 0.00522 | 0.00520 |
| maximum | 0.01175 | 0.00736 | 0.00635 | 0.00436 | 0.00694 | 0.00624 |
| GGAC with Fixed Controllers and Plant Models | | | | | | |
| minimum | 0.00502 | 0.00479 | 0.00415 | 0.00343 | 0.00499 | 0.00517 |
| average | 0.00545 | 0.00567 | 0.00490 | 0.00415 | 0.00558 | 0.00559 |
| maximum | 0.00812 | 0.00627 | 0.00640 | 0.00471 | 0.00648 | 0.00573 |

population members enables the GA to find reasonably good controllers quickly and then search nearby to find better ones.

Table I shows the minimum, average, and maximum errors between the reference input and braking process output for 100 simulations using the GMRAC with fixed population members. Over the course of 100 simulations, the GMRAC with fixed population members had a smaller difference between minimum and maximum errors than did the GMRAC with

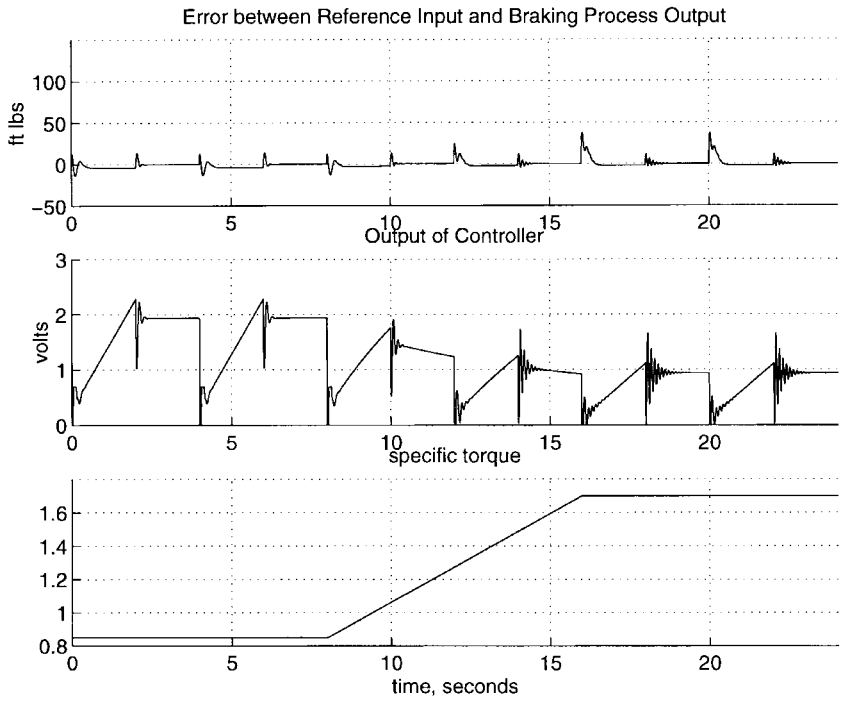



Fig. 10. Results using GMRAC with fixed population members.

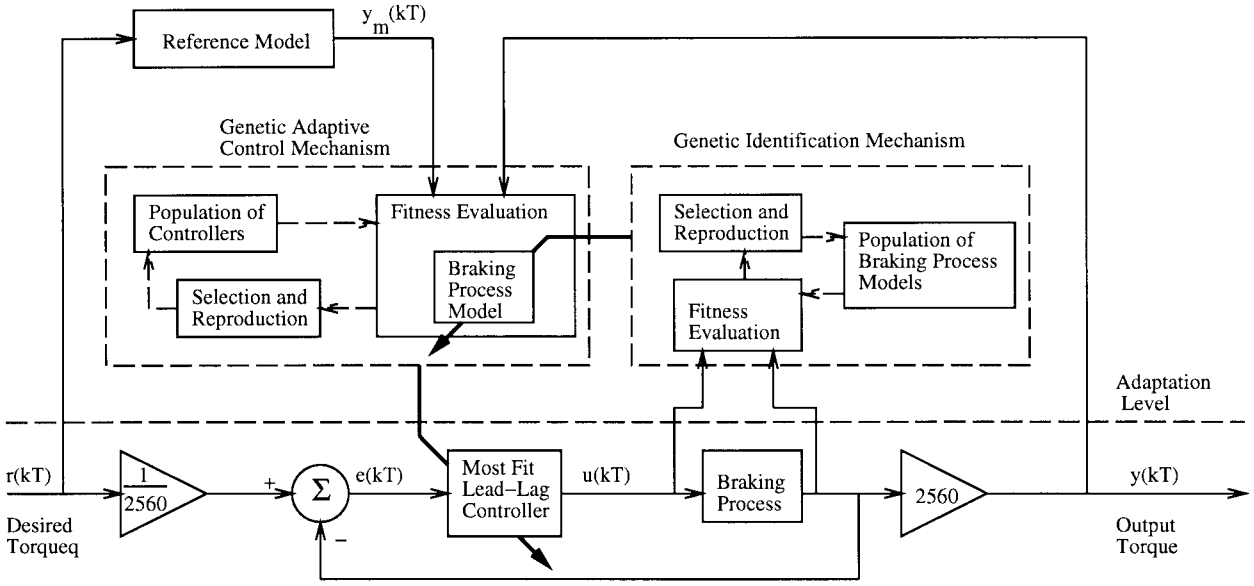



Fig. 11. GGAC for base braking.

no fixed population members. This was expected because the fixed models add a deterministic element to the inherently stochastic genetic algorithm. Fig. 10 shows the results of a typical simulation run. Note that this controller performs very well with little difference in performance as the specific torque increases.

## V. General Genetic Adaptive Control

In this section we expand on the genetic model reference adaptive controller in Section IV, no longer assuming we have a good model of the braking process, but instead use another genetic algorithm to identify the braking process model. This braking process model is then used in the fitness evaluation of the first genetic algorithm which attempts to find a good controller. Fig. 11 shows the GGAC.

### A. GGAC for Base Braking

The plant model structure is similar to the one used in the GMRAC, but a constant offset is assumed. The plant model is defined as

$$\hat{y}(z) = \frac{ku(z) + dz^3}{(z - p_1)(z - p_2)(z - p_3)}. \tag{3}$$

The constant offset $d$ is used to help identify inherent friction in the automobile [as modeled in the brake system by the $F(y)$ function described in Section II].

The second genetic algorithm attempts to identify the parameters $k$, $p_1$, $p_2$, $p_3$, and $d$. The gains are restricted to lie in regions where we know the parameters should be. The gain of the braking process model $k$ was restricted to $0 \leq k < 0.5$. The first two poles (the poles of the braking process) were restricted to $0.9 \leq p_1, p_2 < 1.1$ and the third pole (the pole of the torque sensor) was restricted to $0.6 \leq p_3 < 0.8$. The constant offset was restricted to $0 \leq d < 10^{-5}$. All the parameters are restricted because we assume we know a fair amount of information about the braking system.

The individual process models are defined by a 15-digit chromosome, and each of the five parameters is represented with three digits. The population of process models consists of 100 individuals. The probability of mutation was set to 0.1 (relatively high since we want to make sure that the populations do not stagnate such that the controller will not adapt). The probability of crossover was set to 0.8.

Because the plant parameters do not change very quickly and the processing time is short ($T = 0.005$ s), the identification genetic algorithm only computes a new generation once every five samples. Furthermore, the identification GA is not run for the first ten samples of every ramp-step in the simulation because the braking process input is nearly zero at this time and hence no information can be obtained to help identify the plant parameters.

The fitness function of the genetic identification algorithm is as follows: For each plant model candidate in the population $P_i$ do the following.

1) Initialize the states of the discrete-time process model with the outputs of the actual braking process.
2) Using the past $N = 20$ samples of braking control signal $u(k)$ use the process model candidate to predict the past outputs of the braking process, $\hat{y}_i(k)$. Compute the error between the predicted output $\hat{y}_i(k)$ and the actual output $y(k)$ for each time step.
   For $j = 1$ to $N$

$$\begin{aligned}\hat{y}_i(k+j) = {} & k_i u(k+j-3) \\ & - (-p_{1i} - p_{2i} - p_{3i})\hat{y}_i(k+j-1) \\ & - (p_{1i}p_{2i} + p_{1i}p_{3i} + p_{2i}p_{3i})\hat{y}_i(k+j-2) \\ & - (-p_{1i}p_{2i}p_{3i})\hat{y}_i(k+j-3) + d_i.\end{aligned}$$

   Next $j$
   The braking process model uses the parameters $k_i$, $p_{1i}$, $p_{2i}$, $p_{3i}$, and $d_i$ from the process model candidate, $P_i$. Then

$$\hat{e}_i = \sum_{j=1}^{N} [y(N-j) - \hat{y}_i(N-j)]^2.$$

3) Assign fitness $J_i$ to each plant model candidate

$$J_i = \frac{\alpha}{\hat{e}_i + \alpha}.$$

   Here $\alpha = 0.0001$.
4) Repeat Steps 1)–3) for each member of the population.
5) The maximally fit braking process model becomes the model used for the next time step.

The fitness function is designed to compare how well each plant model tracks the output of the actual braking system, given the actual inputs to the braking system. The error at each time step is computed, squared and summed over the model estimation window, $N$. The plant model with the smallest error sum, $\hat{e}_i$, will have the largest fitness value $J_i$ and will be selected as the model used in the next time step. The value for $\alpha$ was set to 0.0001. In general, we usually select $\alpha$ to be one or two orders of magnitude less than the average error sum, $\hat{e}$, to provide a good mix of fitness values in the population. The model estimation window, $N$ was set to 20 because this provides enough time to observe the braking process model, yet does not require too much controller processor time.

Once the braking process model is determined, it is placed into the fitness function of a genetic algorithm that evolves a good controller. This operates exactly like the GMRAC described previously. In all simulations of the GGAC, we use the GMRAC technique with fixed controllers because we have already demonstrated this improves average tracking performance.

### B. GGAC Results

Fig. 12 shows the results of the braking simulation using the GGAC. The GGAC performs consistently, regardless of the specific torque of the brakes. While the GGAC does not perform as well as the GMRAC, it also assumed less knowledge of the braking process model. The results in Fig. 12 represent the behavior for only one simulation run. Because the GGAC is stochastic, results vary with each simulation. On average, the GGAC greatly outperformed the conventional controller, and outperformed the FMRLC. While the GGAC did not outperform the GMRAC, it required less knowledge of the braking process. See Table I for a summary of these results.

### C. GGAC with Fixed Plant Models

Just as fixed controllers in the GA population can improve the performance of a closed-loop system, so too can fixed plant models. Similar to the work in [14], fixed plant models in a GA population help the GA to quickly identify a reasonable plant model and search nearby for a better one. The parameters of the 16 fixed plant models used in GGAC are defined by all possible combinations of

$$\begin{aligned} k &\in \{0.167, 0.333\} \\ p_1 &\in \{0.967, 1.033\} \\ p_2 &\in \{0.967, 1.033\} \\ p_3 &\in \{0.667, 0.733\} \qquad d = 0. \end{aligned}$$

Table I shows the minimum, average, and maximum errors between the reference input and braking process output for 100 simulations using the GGAC with both fixed plant models and fixed controllers. No simulation plot is shown for the GGAC with fixed plant models because the results on average are very similar to the GGAC without fixed plant models.

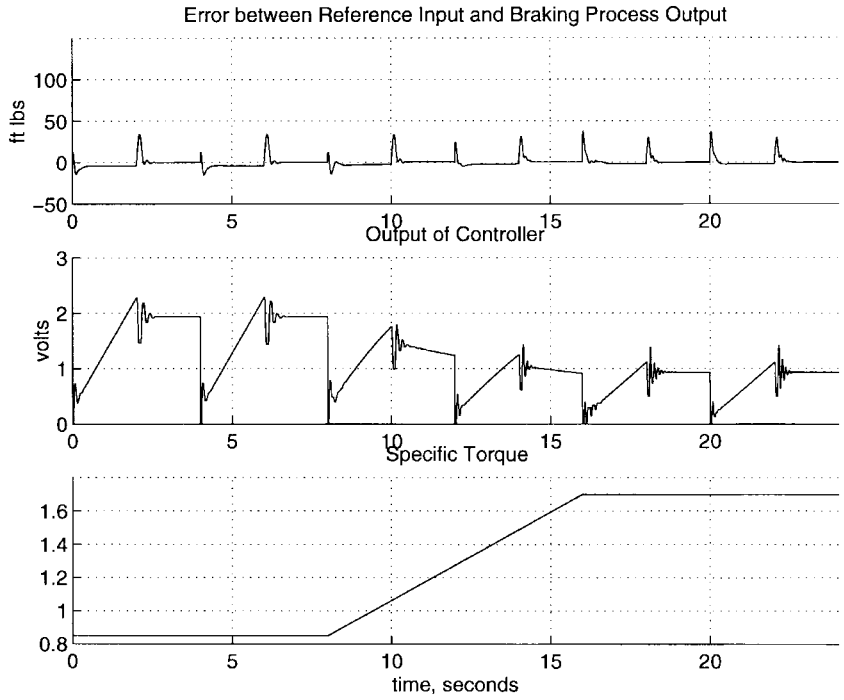



Fig. 12. Results using GGAC with fixed controllers.

The advantage of the GGAC with fixed plant models is its ability to find a reasonable plant model more quickly than the GGAC without fixed plant models. This is critical in the first few seconds of the simulation when very little data from the plant is available. The GGAC without fixed plant models sometimes has difficulties tracking the reference input early in the simulation (as can be seen from the maximum tracking error for the first four seconds in Table I). The GGAC with fixed plant models dramatically reduced this maximum tracking error.

## VI. Discussion

### A. Summary of Results

For all simulations, the error between the reference input and the output of the braking process was computed at each time step. This error was squared and summed for every time step in the simulation to quantify the performance of each controller (note that the amount of control energy used in all cases was comparable and not as important for this application, so we do not use this as an additional performance measure). For the genetic adaptive control techniques, the simulations were run 100 times and the minimum, average, and maximum errors were determined. The results are separated for each 4-s ramp-step input and shown in Table I.

Note that all the intelligent control techniques performed significantly better at tracking the reference input than the conventional lead-lag controller, especially when the specific torque increased. The FMRLC, aside from the first few seconds where it learned to control the braking process, performed quite well. The GMRAC performed well, and the GMRAC with fixed controllers performed the best on average of all the control techniques investigated. Of course the GMRAC also requires the most information about the braking process and uses this information well to track the reference input. The GGAC performed consistently, outperforming the FMRLC, and performing almost as well as the GMRAC while requiring less information about the braking process. The GGAC with both fixed controllers and plant models performed the most consistently, with virtually no difference in the tracking error over the whole range of specific torque values.

We must point out once again, however, that the lead-lag compensator design was for the cold brake condition and in this region (0–4 s) it performs quite well with little oscillation in its control input. Later in the simulations when the brakes heat up the performance of the lead-lag compensator degrades somewhat, but it uses less oscillations in the control input than the intelligent controllers.

### B. Computational Complexity

By far the simplest algorithm to implement is the lead-lag compensator. For it there are five gains to store for its difference equation and hence five multiplications and four additions for each time step. Next, we discuss how to come up with “order of magnitude” approximations of the number of operations needed per step (i.e., within the sampling interval) and memory needed for each of the intelligent control strategies. This analysis is meant to help gain insights into implementation issues for the controllers presented in this paper and to give an idea of what must be paid to get the performance improvements that the intelligent controllers offer.

The complexity of the FMRLC is not too significant since it only has 36 rules in the fuzzy controller and 25 rules in the inverse model. Its update scheme implemented by the

TABLE II
COMPUTATIONAL COMPLEXITY

| | Number of Operations per Step | Memory |
|---|---|---|
| Conventional Lead-Lag | 5 multiplications, 4 additions | 5 gains to store |
| FMRLC | Adaptation: 4 multiplications, 4 sums; Reference model: 6 multiplications, 5 additions; Fuzzy inverse model, fuzzy controller: compute values of input membership functions (which depends on how you code it) and compute outputs (4 multiplications and 8 additions for each assuming that the areas under the implied fuzzy sets are computed). | Adaptation: 28 values; Reference model: 6 values; Fuzzy inverse model and fuzzy controller: 61 output membership function centers, 22 input membership function centers, plus parameters to fully define all membership functions, plus storage of all 61 rules (essentially pointers between membership functions). |
| GMRAC/GGAC | GMRAC has $O = P(P+40N+20C+60)$ operations / step, or for us 4230 operations / step, and there is roughly three times this amount for GGAC. | Reference model: 6 values; Population elements: 8; $N \times P = 80$ values for look-ahead, plus the system model. Roughly twice this amount for GGAC. |

adaptation mechanism is simple and hence computation time is short (only four multiplications and four sums) to update the fuzzy controller. There are six multiplications and five additions for the reference model. The memory needed for the adaptation mechanism depends on the number of steps the mechanism looks back in time to update rules; in this case, it looked back three steps. The mechanism needs to store the degrees of certainty of the four rules that could have been on over the last four steps, and the four past center values for these four rules so $12+16 = 28$ values must be stored. However, this ignores the computations necessary for computing the fuzzy inverse model and fuzzy controller outputs. For this, first note that we must store 61 output membership function centers (36 for the fuzzy controller and 25 for the fuzzy inverse model) and 22 input membership function centers (12 for the fuzzy controller and ten for the inverse model). Next, to find the degree to which an input membership function is on amounts to finding the value of a function (a line), that is the value of the membership function (but you can avoid computing the degree of certainty for each input membership function by some simple range checking on the inputs). However, this has to be done for each of the two inputs for both the fuzzy inverse model and the fuzzy controller. Hence, we have to compute the membership values for all 61 rules in the worse case (of course simple strategies may be used to reduce computations here where for instance, you only compute membership values for the eight rules that are on, four rules for the fuzzy controller and four for the inverse model). Next, the areas under the implied fuzzy sets must be computed, but simple geometry can be used (for the area under a chopped-off triangle) rather than an explicit computation of an integral so this only takes four multiplications and an addition. This completes the necessary computations. Overall, we see that the FMRLC is quite a bit more complex than the lead-lag controller.

To better understand the computation time required of the genetic adaptive controllers, we carefully examined our simulation program (written in the C language) and computed roughly the number of operations required per generation. Using the variables $P$ to represent the population size, $N$ to represent the look-ahead time window steps in the fitness function, and $C$ to represent the length of the chromosome, we arrived at the following equation:

$$O = P(P + 40N + 20C + 60).$$

Here $O$ represents the number of operations per generation (i.e., per time step) of the genetic algorithm, where an operation is any addition, multiplication, subtraction, division, assignment, increment, comparison, or declaration. This equation represents a rough estimate; we were careful to overestimate calculations when simplifying this equation. For memory, you need to store the reference model, the population ($P$ elements), the results of looking ahead $N$ steps for each of these $P$ elements (about $N \times P$ parameters), and the model of the system.

Using this equation, we can see that the GMRAC requires roughly 4230 operations per generation. Using a sampling time of $T = 0.005$ s, that amounts to approximately 850 000 operations per second. This number is less for the case with fixed population members. The GGAC uses two genetic algorithms, with the second GA having a significantly higher population size and chromosome length. The GGAC requires approximately 3.3 million operations per second, assuming a sampling time of $T = 0.005$ s for the genetic adaptive control algorithm and $T = 0.025$ s for the genetic identification algorithm. Of course this computation time could be reduced with more streamlined code and smaller population size and chromosome length. Because this research was in simulation, we did not attempt to optimize the computation time. We are confident that substantial improvements could be made in terms of processing time. Nevertheless, with the cheap and powerful microprocessors widely available today, a controller

that requires 3.3 million operations per second is certainly implementable.

The results of this discussion are summarized in Table II. Overall, our conclusion is that the performance improvements obtained with the intelligent control methods cost us in computational complexity. The cost is not too great for the FMRLC but is significant when we use our GMRAC/GGAC and this demands that we study ways to simplify the code that implements these controllers.

## VII. Concluding Remarks

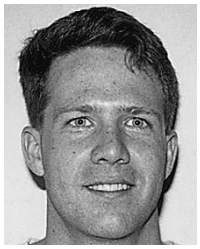
We have used the work from [4] and [5] to define the base braking control problem and have developed two intelligent control methods for this system. Clearly the approaches and conclusions that we present are somewhat preliminary and are in need of further significant investigations. For instance, it would be useful to perform stability, convergence, and robustness analysis of both the FMRLC and GMRAC to help evaluate this safety-critical automotive system. While the model that we use from [4] and [5] has proven to be quite adequate for the development of controllers that have been experimentally evaluated at a test track on a vehicle, it would be valuable to evaluate the developed controllers in the field. This would force us to take a very careful look at the requirements for real-time implementations of the intelligent controllers (our preliminary investigations indicate that we should be able to implement these in real time).

## Acknowledgment

The authors would like to thank Prof. S. Yurkovich for his help in all aspects of this brake system project which have involved significant efforts over many years in modeling and development of conventional controllers for this brake system. They would also like to thank him for his helpful discussions throughout the research on intelligent control methods for the brake system. Also, they would like to thank J. Hurtig for her assistance in establishing the model and simulation testbed for our evaluations and for her past work on controller development and implementation that provided a base-line comparison for the methods developed here. Finally, they would like to acknowledge the support and assistance of D. Littlejohn from the Delphi Chassis Division of General Motors.

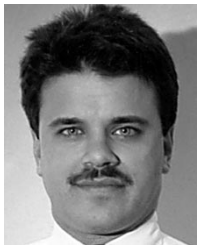
## References

[1] P. Kachroo, "Nonlinear control strategies and vehicle traction control," Ph.D. dissertation, Univ. California, Berkeley, 1993.
[2] S. Drakunov, Ü. Özgüner, P. Dix, and B. Ashrafi, "ABS control using optimum search via sliding modes," *IEEE Trans. Contr. Syst. Technol.*, vol. 3, pp. 79–85, Mar. 1995.
[3] J. Layne, K. Passino, and S. Yurkovich, "Fuzzy learning control for antiskid braking systems," *IEEE Trans. Contr. Syst. Technol.*, vol. 1, pp. 122–129, June 1993.
[4] J. Hurtig, S. Yurkovich, K. Passino, and D. Littlejohn, "Torque regulation with the General Motors ABS VI electric brake system," in *Proc. Amer. Contr. Conf.*, Baltimore, MD, June 1994, pp. 1210–1211.
[5] J. K. Harvey, "Implementation of fixed and self-tuning controllers for wheel torque regulation," Master's thesis, Ohio State Univ., Columbus, 1993.
[6] J. Layne and K. Passino, "Fuzzy model reference learning control for cargo ship steering," *IEEE Contr. Syst. Mag.*, vol. 13, pp. 23–34, Dec. 1993.
[7] L. Porter and K. Passino, "Genetic model reference adaptive control," in *Proc. IEEE Int. Symp. Intell. Contr.*, Columbus, OH, Aug. 1994, pp. 219–224.
[8] W. Lennon and K. Passino, "Intelligent control for brake systems," in *Proc. IEEE Int. Symp. Intell. Contr.*, Monterey, CA, Aug. 1995, pp. 499–504.
[9] W. Lennon and K. Passino, "Genetic adaptive identification and control," 1996, in preparation.
[10] V. Moudgal, W. Kwong, K. Passino, and S. Yurkovich, "Fuzzy learning control for a flexible-link robot," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 199–210, May 1995.
[11] J. Layne and K. Passino, "Fuzzy model reference learning control," in *Proc. 1992 IEEE Conf. Contr. Applicat.*, Dayton, OH, Sept. 1992, pp. 686–691.
[12] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA: Addison-Wesley, 1989.
[13] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolutionary Programs.* Berlin: Springer-Verlag, 1992.
[14] K. S. Narendra and J. Balakrishnan, "Improving transient response of adaptive control systems using multiple models and switching," *IEEE Trans. Automat. Contr.*, vol. 39, pp. 1861–1866, 1994.



**William K. Lennon** received the B.S. degree in electrical engineering in 1994 and the M.S. degree in electrical engineering with a specialization in control systems in 1996, both from the Ohio State University, Columbus.

Presently he is working in the Radio Network Development Department at Ericsson, Research Triangle Park, NC.



**Kevin M. Passino** (S'79–M'90–SM'96) received the Ph.D. degree 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 University, IN, as a Visiting Assistant Professor and is currently an Associate Professor in the Department of Electrical Engineering at the Ohio State University, Columbus.

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 *International 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 Transactions on Fuzzy Systems. He was a Program Chairman for the 8th IEEE International Symposium on Intelligent Control in 1993 and was the General Chair for the 11th IEEE International Symposium on Intelligent Control. He is Coeditor of the book *An Introduction to Intelligent and Autonomous Control* (Boston, MA: Kluwer, 1993), Coauthor of the book *Fuzzy Control* (Reading, MA: Addison-Wesley, 1998), and Coauthor of the book *Stability Analysis of Discrete Event Systems* (New York: Wiley, 1998). His research interests include intelligent systems and control, adaptive systems, stability analysis, and fault-tolerant control.