Journal of Intelligent and Robotic Systems **18:** 209–248, 1997.

Intelligent Control for an Acrobot

SCOTT C. BROWN and KEVIN M. PASSINO*
Department of Electrical Engineering, The Ohio State University, 2015 Neil Avenue, Columbus, OH 43210, U.S.A. e-mail: passino@ee.eng.ohio-state.edu

(Received: 1 November 1996)

Abstract. The acrobot is an underactuated two-link planar robot that mimics the human acrobat who hangs from a bar and tries to swing up to a perfectly balanced upside-down position with his/her hands still on the bar. In this paper we develop intelligent controllers for swing-up and balancing of the acrobot. In particular, we first develop classical, fuzzy, and adaptive fuzzy controllers to balance the acrobot in its inverted unstable equilibrium region. Next, a proportional-derivative (PD) controller with inner-loop partial feedback linearization, a state-feedback, and a fuzzy controller are developed to swing up the acrobot from its stable equilibrium position to the inverted region, where we use a balancing controller to 'catch' and balance it. At the same time, we develop two genetic algorithms for tuning the balancing and swing-up controllers, and show how these can be used to help optimize the performance of the controllers. Overall, this paper provides (i) a case study of the development of a variety of intelligent controllers for a challenging application, (ii) a comparative analysis of intelligent vs. conventional control methods (including the linear quadratic regulator and feedback linearization) for this application, and (iii) a case study of the development of genetic algorithms for off-line computer-aided-design of both conventional and intelligent control systems.

Key words: acrobot, robotics, fuzzy control, genetic algorithms.

1. Introduction

The acrobot, so named because of its similarity to a human acrobat, is an underactuated unstable robot useful as a testbed for studying the theory and application of nonlinear control (see Figure 1). In this paper we apply intelligent control [1, 2] to two challenging robotics control problems associated with the acrobot: swing-up and balancing. To date there seems to be no uniform theory for the control of underactuated robots such as the acrobot, where we try to control the position of its two links with one input. Indeed, its nonlinear dynamics have forced researchers to employ two very different varieties of controllers, one for swing-up and another for balancing. Typically, a heuristic strategy is used for swing-up, where the goal is to force the acrobot to reach its vertical upright position with near zero velocity on both links. Then, when the links are close to the inverted position, a balancing controller is switched on and used to maintain the acrobot in the inverted position (again, see Figure 1).

This paper builds directly on earlier work performed by Professor Mark Spong and his colleagues at the University of Illinois, who have focused on the develop-

* This work has been supported in part by National Science Foundation Grant EEC9315257. Please address correspondence to K. Passino.

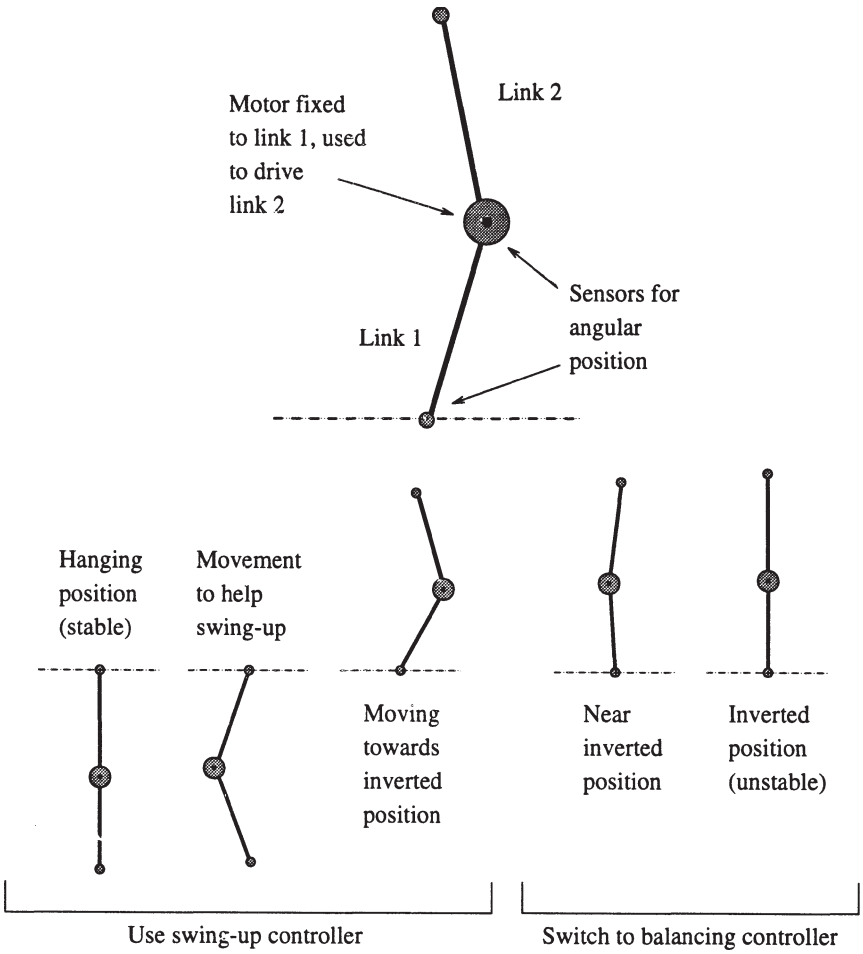

Figure 1. The acrobot.

ment of conventional controllers for the acrobot. Their work in [3] and [4] serves as an excellent introduction to the acrobot and its dynamics. Each paper presents a different partial feedback linearizing controller used for acrobot swing-up. A linear quadratic regulator (LQR) balancing controller is developed for the acrobot in both of these works. The dynamics of a simple acrobot are also stated in both works, however a more complete development of the acrobot dynamics may be found in Spong and Vidyasagar's book on robotics [5]. Two other relevant papers which discuss the acrobot dynamics, the unstable equilibrium manifold, and controllability issues include [6] and [7]. In this paper we study a different type of acrobot than in [3, 4, 6, 7], where we assume that (i) the actuator (motor) used to drive the second link is attached to the end of the first link, (ii) the second link is not allowed to move in a full circle but is constrained so that it cannot

cross over the first link, and (iii) we put a saturation nonlinearity on the torque input from the motor.

We make extensive use of direct and adaptive fuzzy controllers throughout this work. Publications in this area abound in the literature; however, several sources for fuzzy control theory include [8], [9], and [10]. The fuzzy model reference learning controller (FMRLC), an adaptive fuzzy controller used here for the acrobot (see Section 3), has also been applied to many other applications [11, 12]. This paper presents what seems to be the first results on the use of direct and adaptive fuzzy control for the acrobot.

Genetic algorithms (GAs) are also used throughout this work. The interested reader is encouraged to consult [13] and [14] for an introduction to the genetic algorithm (due to space constraints we do not overview the basic mechanics of GAs and simply assume that the reader understands these). Genetic algorithms have in the past been used in the design of conventional and intelligent control systems. For instance, see [15, 16, 17, 18, 19, 20, 21, 22]. In this paper we present what seems to be the first results on the development of GAs for computer-aided-design of conventional and intelligent swing-up and balancing controllers for the acrobot.

This paper is organized into five sections. Section 2 develops a model for the acrobot, including the parameter values to model a realistic physical acrobot. In Section 3 we develop linear and non-linear balancing controllers for the acrobot. The disturbance rejection capabilities of each balancing controller are investigated. We also discuss how we use genetic algorithms to tune the balancing controller parameters, and present the results of these re-tuned controllers. In Section 4 we develop three types of swing-up controllers; genetic algorithms are then used to tune the swing-up controller parameters. Lastly, Section 5 provides a brief summary of the results.

2. The Acrobot System and Dynamics

The acrobot has a single actuator at the elbow and no actuator at the shoulder; the system is underactuated because we desire to control two links of the acrobot (each with one degree of freedom) with only a single system input. The configuration of a simple acrobot, from which the system dynamics are obtained, is shown in Figure 2. The joint angles q_1 and q_2 serve as the generalized system coordinates; m_1 and m_2 specify the mass of the links; l_1 and l_2 specify the link lengths; l_{c1} and l_{c2} specify the distance from the axis of rotation of each link to its center of mass; and lastly, I_1 and I_2 specify the moment of inertia of each link taken about an axis coming out of the page and passing through its center of mass. The single system input, τ, is defined such that a positive torque causes q_2 to increase (move in the counter-clockwise direction).

In [5], Spong has developed the equations of motion of a planar elbow manipulator; this manipulator is identical to the acrobot shown in Figure 2, except that

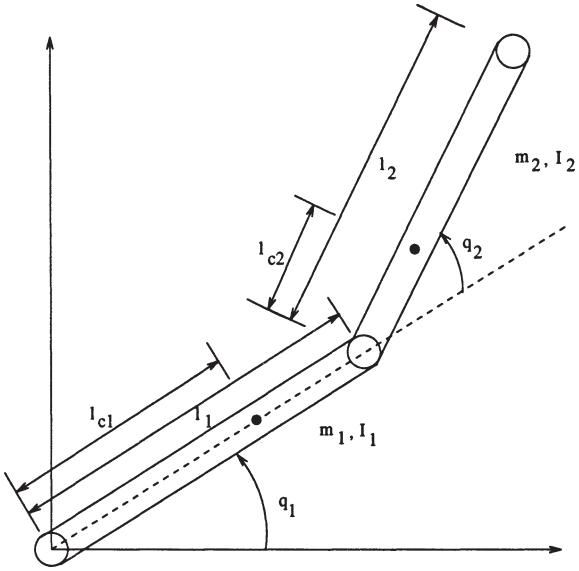

Figure 2. Simple acrobot notation.

it is actuated at joints one *and* two. The dynamics of the acrobot are simply those of the planar manipulator, with the term corresponding to the input torque at the first joint set equal to zero and are given by

$$d_{11}\ddot{q}_1 + d_{12}\ddot{q}_2 + h_1 + \phi_1 = 0, \tag{1}$$

$$d_{12}\ddot{q}_1 + d_{22}\ddot{q}_2 + h_2 + \phi_2 = \tau, \tag{2}$$

where the coefficients in equations 1 and 2 are defined as

$$d_{11} = m_1 l_{c1}^2 + m_2(l_1^2 + l_{c2}^2 + 2l_1 l_{c2}\cos(q_2)) + I_1 + I_2, \tag{3}$$

$$d_{22} = m_2 l_{c2}^2 + I_2, \tag{4}$$

$$d_{12} = m_2(l_{c2}^2 + l_1 l_{c2}\cos(q_2)) + I_2, \tag{5}$$

$$h_1 = -m_2 l_1 l_{c2}\sin(q_2)\dot{q}_2^2 - 2m_2 l_1 l_{c2}\sin(q_2)\dot{q}_2\dot{q}_1, \tag{6}$$

$$h_2 = m_2 l_1 l_{c2}\sin(q_2)\dot{q}_1^2, \tag{7}$$

$$\phi_1 = (m_1 l_{c1} + m_2 l_1)g\cos(q_1) + m_2 l_{c2} g\cos(q_1 + q_2), \tag{8}$$

$$\phi_2 = m_2 l_{c2} g\cos(q_1 + q_2). \tag{9}$$

In our acrobot model we have also limited the range for joint angle q_2 to $[-\pi, \pi]$ (i.e., the second link is not free to rotate in a complete revolution – it

Table I. Acrobot model parameters used in simulations

Parameter	Value
m_1	1.9008 kg
m_2	0.7175 kg
l_1	0.2 m
l_2	0.2 m
l_{c1}	1.8522×10^{-1} m
l_{c2}	6.2052×10^{-2} m
I_1	4.3399×10^{-3} kg·m^2
I_2	5.2285×10^{-3} kg·m^2

cannot cross over the first link). We have also cascaded a saturation nonlinearity between the controller output and plant input to limit the input torque magnitude to 4.5 N/m. This seemed a reasonable torque limitation based upon current motor designs. The model parameter values which we have used throughout this paper have been chosen as realistically as possible. In choosing these parameters, we have assumed that the input torque at joint two would be supplied by a motor mounted at the *end* of link one. With this assumption, the acrobot parameters that we have chosen are summarized in Table I.

For all of the simulations performed in this paper we use a fourth-order Runge–Kutta technique with an integration step-size of 0.0025 seconds. To simulate the effects of implementing the controllers on a digital computer we sample the output signals with a period $T_s = 0.01$ seconds, and only update the control input every T_s seconds (holding the value constant in between updates).

3. Balancing Controllers

The first controllers developed for the acrobot were balancing controllers. Balancing the acrobot refers to maintaining the acrobot in the inverted position when it starts close to this position. To balance the acrobot in the inverted position, one must design a controller which stabilizes the behavior of the system in some region about an equilibrium point. The acrobot has a single stable equilibrium point corresponding to both links hanging vertically *beneath* joint one. Rather than a single inverted equilibrium point, however, the acrobot dynamics result in a manifold of inverted equilibrium positions. Physically, the acrobot is in an inverted equilibrium position whenever the system center of mass is directly above joint one [7]. Each equilibrium position is associated with a unique constant torque input [6]; thus, only the completely vertical inverted position results in a zero torque input. Several of the infinitely many inverted equilibrium positions along the equilibrium manifold are shown in Figure 3. In this section we will design controllers to balance the acrobot only in the completely verti-

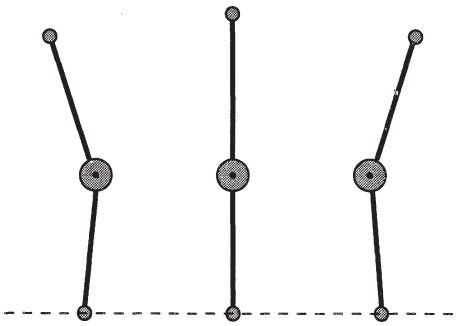

Figure 3. Several possible equilibrium positions on the equilibrium manifold.

cal position. Other works have designed controllers to move the acrobot along the equilibrium manifold (see [7] and [6]). Indeed, Hauser and Murray showed in [7] that the linearized acrobot dynamics are completely controllable along the equilibrium manifold and can be controlled locally by linear controllers.

In this section, both linear and nonlinear balancing controllers have been developed for the acrobot. A linear quadratic regulator (LQR) is the first balancing controller introduced. Next, two nonlinear controllers are introduced: a direct fuzzy controller, and a fuzzy model reference learning controller (FMRLC) [11]. This section will develop all of the balancing controllers and also provide simulation results demonstrating their performance on the nominal plant and the plant with constant and periodic torque disturbances. Genetic algorithms will then be used to tune the controller parameters for these balancing controllers.

3.1. LINEAR QUADRATIC REGULATOR

The first step in developing any linear controller is obtaining a linear model of the system. A linear model of the acrobot was obtained by linearizing the acrobot dynamics about the inverted position ($q_1 = \pi/2$, $q_2 = 0$, $\dot{q}_1 = 0$, $\dot{q}_2 = 0$) with $\tau = 0$. Defining the state vector $\mathbf{x} = [q_1 - \pi/2,\ q_2,\ \dot{q}_1,\ \dot{q}_2]^T$ transforms the balancing control problem to a regulation problem. The acrobot dynamics linearized about $\mathbf{x} = [0,\ 0,\ 0,\ 0]^T$ may be described by

$$\dot{\mathbf{x}} = A\mathbf{x} + B\tau, \tag{10}$$

$$\mathbf{y} = C\mathbf{x} + D\tau. \tag{11}$$

The numerical values for the A, B, C, and D matrices for the acrobot model developed in Section 2 were determined to be

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 49.4782 & -5.5038 & 0 & 0 \\ -50.0109 & 66.2336 & 0 & 0 \end{bmatrix}, \tag{12}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ -23.9348 \\ 175.7326 \end{bmatrix}, \tag{13}$$

$$C = I_{4\times 4}, \tag{14}$$

$$D = 0_{4\times 1}. \tag{15}$$

The poles of the linearized open-loop system are found to be at -8.7431, 8.7431, -6.2666, and 6.2666, showing that the inverted equilibrium position is (as expected) unstable.

The linear quadratic regulator (LQR) is an optimal state-feedback controller that minimizes the quadratic cost criterion

$$J = \int_0^\infty (\mathbf{x}^T Q \mathbf{x} + \tau^T R \tau)\, \mathrm{d}t. \tag{16}$$

The matrices Q and R in Equation (16) are used to weight the states and input in the cost criterion. The LQR designed for the linearized acrobot system has weighting matrices given by

$$Q = \begin{bmatrix} 1000 & -500 & 0 & 0 \\ -500 & 1000 & 0 & 0 \\ 0 & 0 & 1000 & -500 \\ 0 & 0 & -500 & 1000 \end{bmatrix}, \tag{17}$$

$$R = [10000]. \tag{18}$$

Matlab was used to determine the corresponding static state-feedback gains given by

$$K_{\mathrm{LQR}} = [-113.8908, \quad -9.5070, \quad -17.3951, \quad -1.9405]. \tag{19}$$

The weighting matrices chosen are similar to those used by Spong in [3] for designing a balancing controller for a different acrobot. He used the same Q matrix and a different value for R. A larger value of R was used here for two reasons. First, increasing the value of R places greater emphasis on reducing the torque input requirements in the cost criterion. Second, while increasing the value of R does tend to result in a slightly worse balancing performance (because the control input is reduced), a lower R value results in a single fast closed-loop pole that can cause problems with discrete implementation. The linearized system with the LQR designed above has closed-loop poles at -60.9287, -1.5402, $-6.4372 - 0.1331j$, and $-6.4372 + 0.1331j$. Other controllers designed with different Q and R matrices were also tried but found to be less successful.

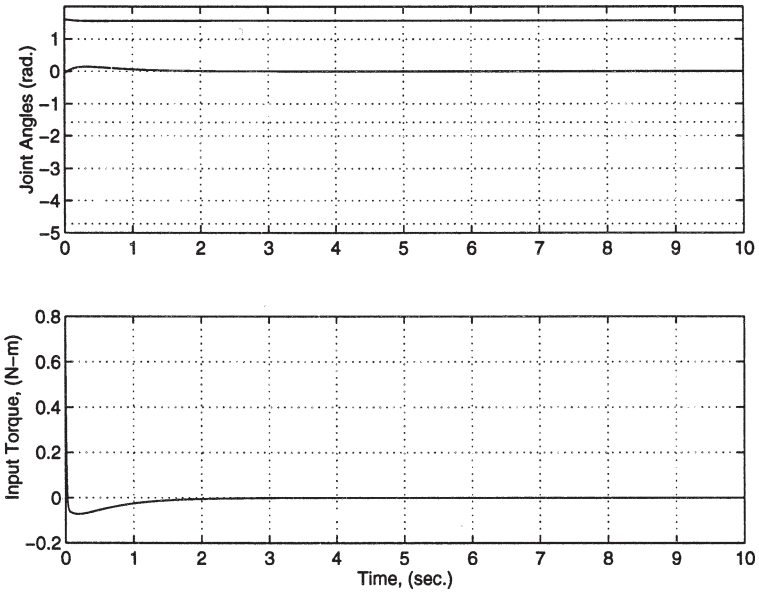

Figure 4. Simulation results for acrobot with LQR.

The initial condition used to test this and other balancing controllers was: $q_1 = \pi/2 + 0.04$, $q_2 = -0.0500$, $\dot{q}_1 = -0.2000$, and $\dot{q}_2 = 0.0400$. This arbitrarily chosen initial condition is such that the first link is approximately 2.29° beyond the inverted position, while the second link is displaced approximately $-2.86°$ from the first link. The initial velocities are such that the first link is moving away from the inverted position, while the second link is moving into line with the first link. The simulation results for the acrobot with this initial condition and the LQR are shown in Figure 4. The optimal LQR is successful at balancing the acrobot in the inverted position; the controller performance for this initial condition is also quite similar to the performance found with other initial conditions provided they are not too far from the balanced position.

3.2. MANUALLY TUNED NONLINEAR CONTROL

In this section we will develop two nonlinear controllers, a direct fuzzy controller and a FMRLC. Because the acrobot is a nonlinear system we might expect the nonlinear controllers to have some advantage over the LQR. These controllers will be tested for the same set of initial conditions as the LQR, and their performance compared with the performance of the LQR.

3.2.1. *Direct Fuzzy Control*

A direct fuzzy controller was the next balancing controller designed for the acrobot. The fuzzy system used has four inputs: $g_0(q_1 - \pi/2)$, $g_1 q_2$, $g_2 \dot{q}_1$, and

$g_3\dot{q}_2$; and a single output τ (each input is preceded by a static normalizing gain term, g_0, g_1, g_2, and g_3, while the output is multiplied by a static gain, h). The 'effective universe of discourse' (i.e., the range where the membership functions are not saturated at one) for each input and the output is $[-1, 1]$. We first attempted to choose the rule-base intuitively and determine values for the fuzzy controller gains based upon our understanding of the system. The system was simulated with different initial conditions and input torques in an attempt to ascertain typical operating ranges for the system states and the effect of different input torques on these states. However, no fuzzy controller gains and rule-bases determined in this manner were successful at balancing the acrobot in the inverted position.

The next method tried for determining the gains g_0, g_1, g_2, g_3, and h uses the LQR gains previously designed for a linearized model of the system. To use this method, one of the input gains, g_i, is first chosen to map the normal operating range of state i to the normalized universe of discourse, $[-1, 1]$. The index i is chosen to correspond to the most important state, i.e., the state most critical in achieving the control objective. With g_i determined, we solve for h so that the 'effective gain*' for the ith input to the fuzzy controller is equal to the the corresponding state-feedback gain for the ith state; that is

$$h = \frac{K_{\mathrm{LQR}}(i)}{g_i}. \tag{20}$$

With h now determined, the remaining gains, g_j ($j = 1, \ldots, 3;\ j \neq i$), are chosen to also match the corresponding state-feedback gains. Using this method,the final fuzzy system gains are related to the state-feedback gains with the relationship

$$K_{\mathrm{LQR}} = [g_0 h,\ g_1 h,\ g_2 h,\ g_3 h]. \tag{21}$$

The rule-base for a fuzzy system with its gains determined in this manner should implement index adding, to emulate the summing effect of a state-feedback controller. Such a scheme is explained in more detail in [23] and [24] for two other applications.

Using the above method in conjunction with the state-feedback gains developed in Section 3.1, a successful direct fuzzy balancing controller was found. The angular position of joint one, q_1, was determined to be the critical state in balancing the acrobot. A gain of g_0 = 10 was chosen as the input normalizing gain for the first fuzzy controller input; this choice for g_0 implies that values of q_1 in the range $[-\frac{1}{10}, \frac{1}{10}]$ radians will be mapped to fuzzy sets without saturation.

* Since the fuzzy controller is a static nonlinear map that is Lipschitz continuous, if we make small changes to its inputs its outputs will only change slightly so that *locally* the fuzzy controller will act like a linear controller; to characterize this we use the term 'effective gain'.

A better fuzzy controller, however, was designed using the same gain selection method in conjunction with another set of state-feedback gains. The state-feedback gains resulting in improved fuzzy controller performance were determined from an LQR designed with a different R weighting value; a value of $R = 1000$ was used instead of the previous value of $R = 10000$. This value change results in state-feedback gains given by

$$K_{\text{LQR}} = [-310.6372, \quad -26.3246, \quad -47.5231, \quad 5.3165], \tag{22}$$

and the corresponding linearized system closed-loop pole locations* at -189.2311, -106.8802, $-6.4360-0.1336j$, $-6.4360+0.1336j$. After some simulation trials, a value of $g_0 = 5.555\bar{5}$ ($1/0.18$ radians) was found to yield the best simulation results. Using the above procedure, the other resulting fuzzy controller gains are given by $g_1 = 0.4708$, $g_2 = 0.8500$, $g_3 = 0.0951$, and $h = 55.9147$.

The best fuzzy system is implemented with 13 triangular membership functions on each universe of discourse, with 50% overlap, evenly distributed over each input universe of discourse. The leftmost and rightmost membership functions are extended to $-\infty$ and $+\infty$, respectively, and there are 49 triangular membership functions distributed over the output universe of discourse. The addition of more membership functions on the input or output universes of discourse did not seem to significantly improve performance. The output membership function centers are mapped by the function

$$C_i = \frac{\text{sign}(i)|i|^z}{(\max(i))^z}, \tag{23}$$

where i is the membership function's linguistic index ranging from $-\max(i)$ to $\max(i)$ (i.e., we number the membership functions on each universe of discourse with integer indices i, where

$$i \in \{\ldots, \ -4, \ -3, \ -1, \ 0, \ 1, \ 2, \ 3, \ 4, \ \ldots\},$$

with '0' labeling the membership function that is centered at zero (these are sometimes called 'linguistic-numeric indices') [8]). The best value for the variable z was found to be 0.8. This choice of z causes the spacing between output membership function centers to decrease as the linguistic index is increased; thus, the output membership functions are spread out most about zero, increasing the controller's effective gain in this region. The fuzzy system's rule-base implements 'index-adding' with saturation (see [23]); min is used for the premise and implication, and center-of-gravity (COG) is used for defuzzification [8].

Simulation results for this fuzzy controller are shown in Figure 5. These results were obtained using the same initial conditions used in Section 3.1. When compared with the LQR simulation results for the same initial conditions in Figure 4, we see that the LQR controller results in smoother state trajectories

* Note how the reduced value of R has resulted in a faster closed-loop pole.

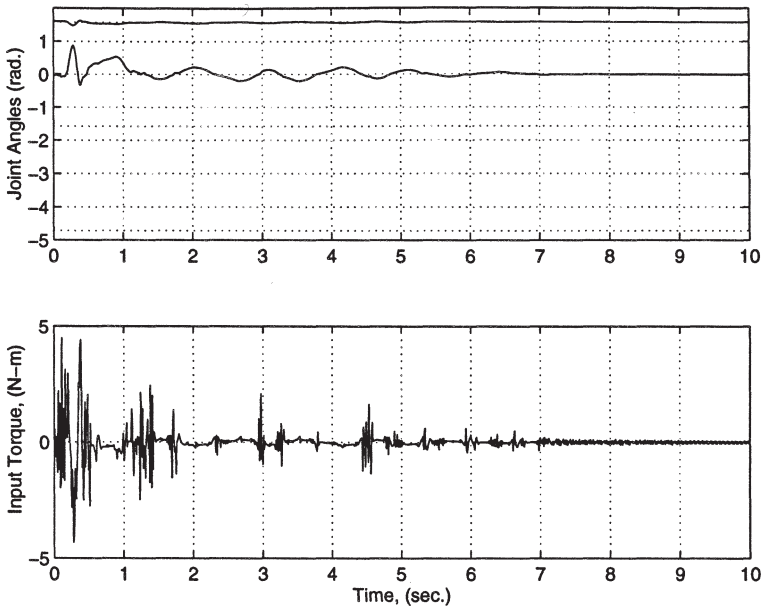

Figure 5. Simulation results for direct fuzzy controller.

and a reduced control input. Clearly, however, this is not due to a deficiency in the fuzzy control approach. Additional tuning would result in performance similar to the LQR case since, theoretically, it is possible to tune the fuzzy controller to implement the same control surface as the LQR (at least locally). The problem is that it is quite difficult to tune the direct fuzzy controller; additionally, more membership functions may be required. This is why we turn to the adaptive fuzzy controller studied next, where we seek a method that will automatically tune the membership functions.

3.2.2. *Fuzzy Model Reference Learning Control*

We saw in the previous section that the fuzzy controller may be manually tuned to achieve reasonable performance. However, different initial conditions can severely degrade the controller's performance. To this end we are motivated to pursue an adaptive strategy for balancing the acrobot in the inverted position.

The FMRLC strategy utilizes a second fuzzy system to modify the output membership centers of the first direct fuzzy controller. In a typical FMRLC configuration, a reference model output is compared with the actual plant output and is used to generate error inputs to the second fuzzy system. A 'fuzzy inverse model' (the rule-base of the second fuzzy system) is then used to generate a desired change in the output membership function centers of the rules that caused the deviation from the reference model. Often the rules and corresponding output membership functions which caused the deviation from the

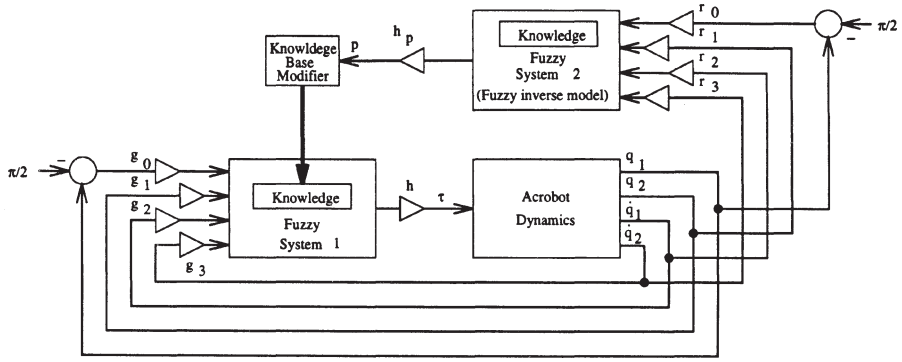

Figure 6. Configuration of balancing FMRLC.

reference model occurred several sampling periods prior to the current output. Due to space constraints we must omit the details of the FMRLC operation and refer the reader to [25, 11] for a more detailed tutorial introduction and several studies of its application.

In the case of the acrobot FMRLC, no explicit dynamical reference model is used. Instead, the plant outputs are multiplied with normalizing gains and used directly as inputs to the second fuzzy system, as shown in Figure 6. Here, the input normalizing gains, r_0, r_1, r_2, and r_3, are set equal to the input normalizing gains used for the direct fuzzy controller, g_0, g_1, g_2, and g_3, respectively (for justification of this choice see [11]). Each input to the second fuzzy system has 11 triangular membership functions, with 50% overlap, evenly distributed over its respective universe of discourse. The output universe of discourse is spanned by 41 triangular membership functions. The membership function centers are mapped with the same type of I/O mapping used for the direct fuzzy controller, but here a value of $z = 1.3$ has been used. The rule-base of the second fuzzy system is once again implemented by adding the indices (see the previous section for more details). The output of this fuzzy system is passed through a gain $h_p = 0.5$; this value affects the 'learning rate' of the FMRLC. The larger the value of h_p, the more the affected membership function centers are changed, and the faster the system adapts (of course stability concerns limit how high h_p can be). For more details on design procedures and tuning of the FMRLC see [11, 23].

The knowledge-base modifier actually changes the output membership function centers of the direct fuzzy controller (Fuzzy System 1 in Figure 6). The direct fuzzy controller has been augmented to store the rules that were on, and the corresponding degree of membership in each output membership function, for the past d sampling periods. The knowledge-base modifier uses this stored information to change the center of each membership function which had a nonzero degree of membership d samples ago. Each effected membership function center,

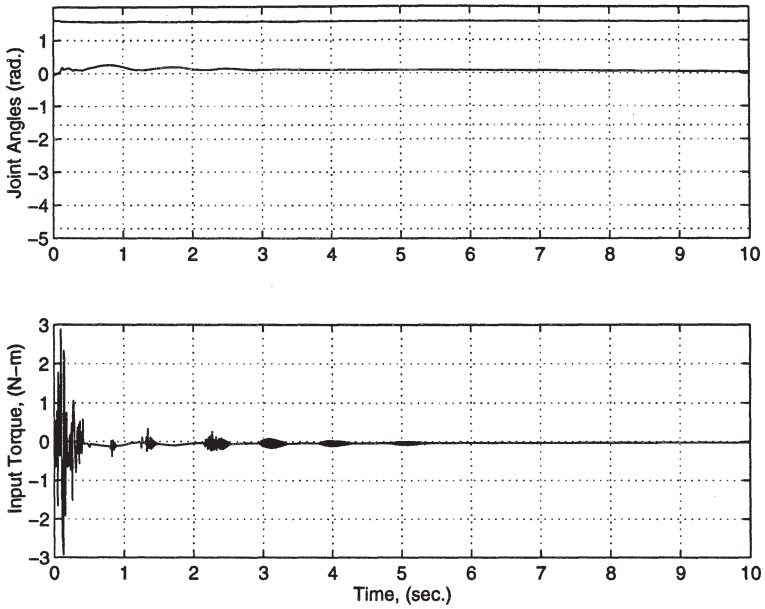

Figure 7. Simulation results for acrobot with FMRLC.

$C_{i,j,k,l}((k-d)\cdot T_s)$, is shifted by the relationship

$$C_{i,j,k,l}((k-d)T_s) := C_{i,j,k,l}((k-d)T_s) + \\ + p(kT_s)\cdot \mu_{i,j,k,l}((k-d)T_s) \tag{24}$$

at each $t = kT_s$. The subscripts i, j, k, l on C and μ represent the center and degree of membership for the membership function corresponding to the rule with linguistic indices i, j, k, l for each input, respectively.

Often when FMRLC is used, the direct fuzzy controller is initialized with no knowledge, i.e., all output membership function centers are set to zero (see, e.g., [11]). In this case, all of its subsequent knowledge is learned via the FMRLC. For the acrobot system, however, it seems this is not possible because the inverted position is an unstable equilibrium point. Without some initial knowledge, the acrobot falls over before the controller has time to learn. Thus, the direct fuzzy controller is initialized with the same knowledge base as the direct fuzzy controller used in Section 3.2.1.

The response of the acrobot with the FMRLC for the initial condition used previously is shown in Figure 7. A value of $d = 1$ was used for this simulation. For the acrobot, a single sample delay was found to yield the best simulation results for all of the initial conditions tried. Comparing the response in Figure 7 with the response of the direct fuzzy controller shown in Figure 5, we see a dramatic improvement for the FMRLC over the direct fuzzy controller. However, the response of the system with FMRLC is not quite as smooth as the system

with the LQR (see Figure 4). The FMRLC was found to perform similarly for other initial conditions; hence, the FMRLC significantly reduces the dependence on initial conditions observed with the direct fuzzy controller. In the next section we will also show that the FMRLC seems to have certain advantages over the LQR for disturbance rejection.

3.3. CONTROLLER DISTURBANCE REJECTION

In this section we will consider the disturbance rejection abilities of the three balancing controllers already developed. The first type of disturbance studied was a constant disturbance at the torque input, which could result from a small offset error in the power amplifier used to drive the acrobot motor. A periodic disturbance at the torque input was studied next; such a disturbance could be a result of noise emanating from surrounding electrical equipment or the AC power lines. To demonstrate the behavior of the system for another initial condition, the simulation results shown in this section were started from the initial condition: $q_1 = 1.4704$, $q_2 = 0.0828$, $\dot{q}_1 = 0.5811$, and $\dot{q}_2 = -0.2020$. This initial condition implies that the first link is approximately 5.75° from the inverted position, while the second link is displaced approximately 4.74° from the first link (see Figure 2). The initial velocities are such that the first link is moving toward the inverted position, while the second link is moving into line with the first link.

3.3.1. *Constant Torque Disturbance*

The constant disturbance tested was a constant offset added to the plant's torque input. This disturbance had the form

$$\tau = \tau_{\text{desired}} + C. \tag{25}$$

A value of $C = 0.5$ was used in the simulations. The simulation response for the acrobot with the LQR is shown in Figure 8. Here we see that the constant torque disturbance has caused the LQR to balance the acrobot in an equilibrium position other than the completely vertical position on the equilibrium manifold. We omit the plot for the direct fuzzy controller since it results in an unstable system. Figure 9 shows the simulation response for the FMRLC. Not only has the addition of the FMRLC stabilized the system, but it has also compensated for the constant disturbance. Indeed, in approximately 15 seconds, the acrobot has been returned to the completely vertical equilibrium position (performing better than the LQR).

3.3.2. *Periodic Torque Disturbance*

The periodic disturbance tested was a sinusoid added to the plant's torque input. The actual disturbance used in simulation is given by

$$\tau = \tau_{\text{desired}} + 0.3\sin(2\pi t). \tag{26}$$

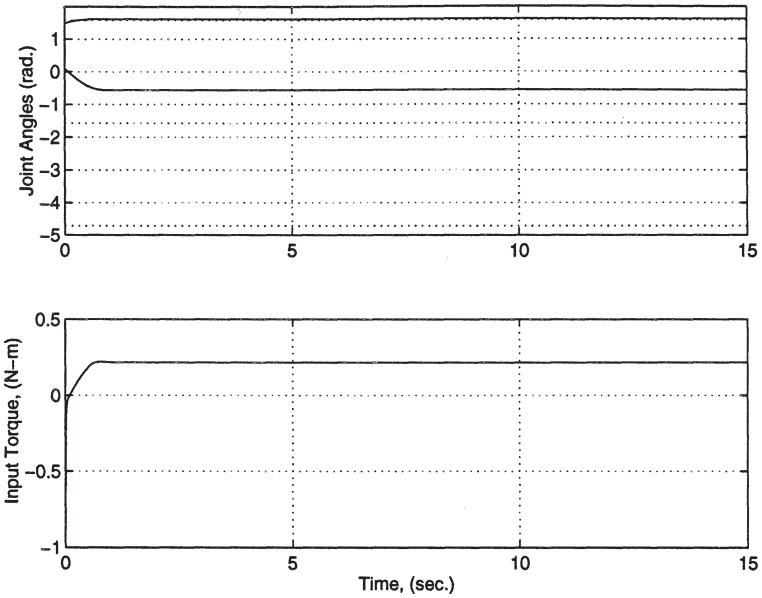

Figure 8. Simulation results for acrobot with LQR and constant torque disturbance.

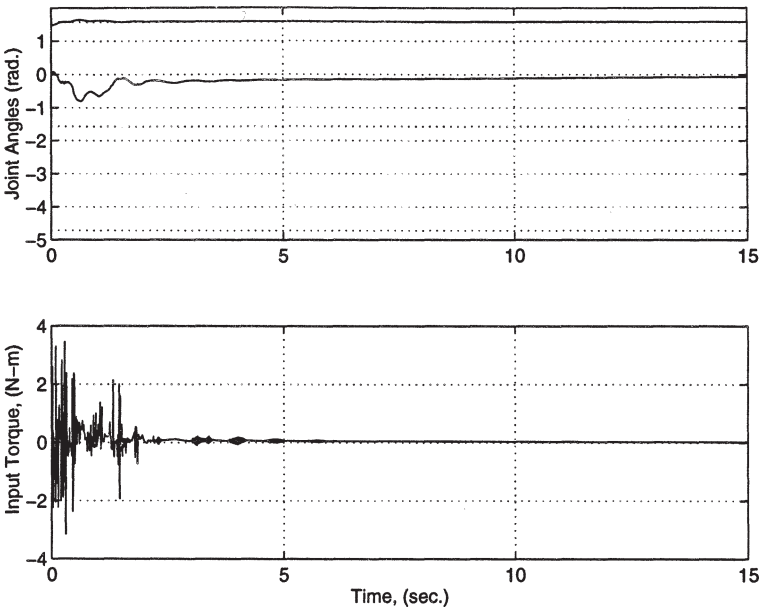

Figure 9. Simulation results for acrobot with FMRLC and constant torque disturbance.

Figure 10 shows the simulation response for the acrobot with the LQR. The results reveal that while the acrobot remained balanced, the disturbance has been passed through to the system with no attenuation whatsoever. The simulation

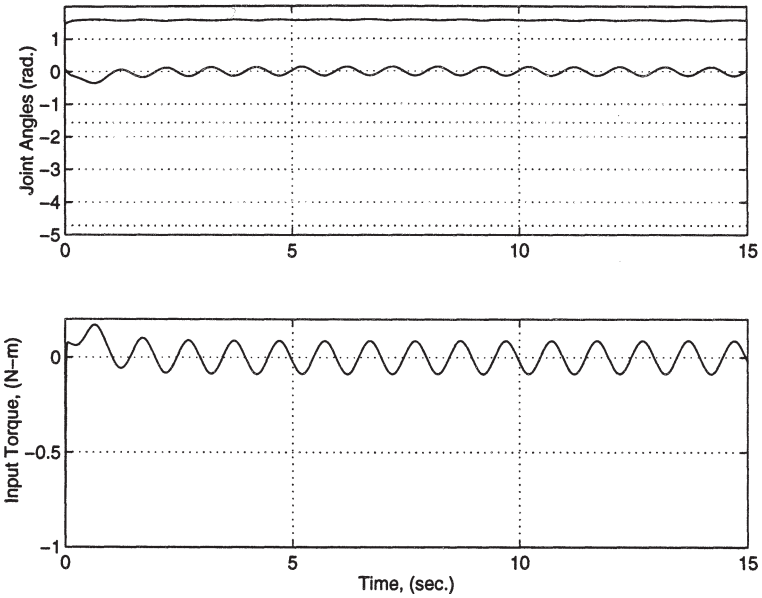

Figure 10. Simulation results for acrobot with LQR and periodic torque disturbance.

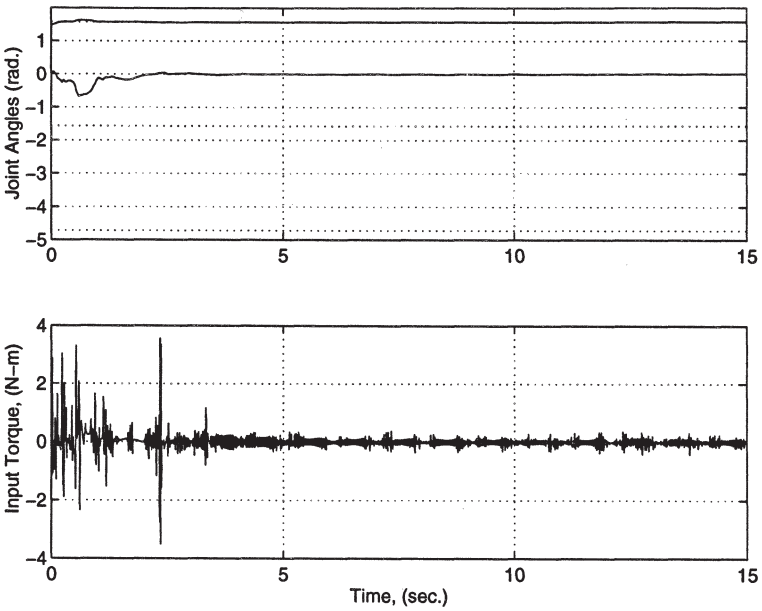

Figure 11. Simulation results for acrobot with FMRLC and periodic torque disturbance.

results for the direct fuzzy controller reveal an even worse response as again the disturbance is not attenuated and its response is not as smooth as that of the LQR. The FMRLC, however, has dramatically outperformed both the LQR

and the direct fuzzy controller. The simulation results shown in Figure 11 reveal that the FMRLC has almost completely eliminated the effects of the disturbance. Indeed, the only real difference between this result and the result obtained with no disturbance is a small high-frequency oscillation on the input to the plant, which seems reasonable.

3.4. BALANCING CONTROLLER TUNING BY A GENETIC ALGORITHM

In this section the parameters of the manually tuned balancing controllers introduced in Section 3.2 will be tuned by genetic algorithms (GAs). In fact, we will show how the GAs are able to successfully improve the performance of each balancing controller for a given initial condition*. Note that in the GA we have a population (set) of chromosomes with each chromosome representing the candidate controller (i.e., the controller parameters are sets of 'genes' called traits and these are represented with standard base-10 numbers). There is a fitness function that is used to evaluate the fitness of each controller in the population based on standard closed-loop control system performance measures (which are carefully quantified below). The GA operates by selecting chromosomes (controllers) to enter a mating pool more often if they have an above average fitness (i.e., have better closed-loop responses). The chromosomes (controllers) in the mating pool are then 'crossed-over' and 'mutated' [13, 14] with certain probablities to form the new generation (this achieves an interpolation between the controllers in the population that are the best and these better controllers are placed in the next generation). Also, we use 'elitism' [13] so that the most fit chromosome (i.e., the best controller) is passed on to the next generation without modification (this helps to ensure that there will be at least one good controller in each generation). Repeating this process results in an evolution of controller parameters and hence provides us with a way to tune our controllers. For more details see [13, 14].

3.4.1. *The Balancing Fitness Function*

Before any GA can be used to tune the controller parameter values, a quantitative measure of the balancing performance is needed; the fitness function provides us with this measurement. The fitness function used to evaluate the performance of the balancing controllers attempts to penalize the movement of the acrobot's two joints, as well as the required control input. The balancing controllers were each simulated for 10 seconds, beginning from the same initial condition. During each run the following statistics were gathered to characterize the controller's performance: the mean of the joint positions, $q_1 - \pi/2$ and q_2, and control input, τ, for $t = 5$ to $t = 10$ seconds (denoted m_{q_1}, m_{q_2}, and m_τ); the normalized sum of the squares of the joint positions and control input for $t = 0$ to $t = 5$ seconds

* The GAs used throughout this paper have been implemented in C using a modified version of a GA program written by Will Lennon.

(denoted s_{q_1}, s_{q_2}, and s_τ); and the standard deviation of the joint positions and control input for $t = 5$ to $t = 10$ seconds (denoted σ_{q_1}, σ_{q_2}, and σ_τ). Using these variables, the balancing fitness function may be written as*

$$f_b = \begin{cases} \mathbf{w}^T\mathbf{s} & \text{if } (\mathbf{w}^T\mathbf{s} > 0.001), \\ 0.001 & \text{else}, \end{cases} \tag{27}$$

where

$$\mathbf{w} = [w_0,\ w_1, \ldots, w_7,\ w_8]^T \tag{28}$$

is a user-specified vector of weights for the vector

$$\mathbf{s} = [|m_{q_1}|,\ |m_{q_2}|,\ |m_\tau|,\ s_{q_1},\ s_{q_2},\ s_\tau,\ \sigma_{q_1},\ \sigma_{q_2},\ \sigma_\tau]^T. \tag{29}$$

Basically, the GA tries to evolve a population of controllers that minimizes f_b (by maximizing $1/f_b$) and thereby reduces variations in the joint angle and control input. By using different values for the weighting vector **w**, it is possible to have a GA tune the balancing controllers to meet different criteria. The sum of the squares have been used as performance measures in the first 5 seconds of simulation to penalize movement and control input during the transition period of the simulation. The standard deviation was used instead of the sum of the squares in the latter simulation period to penalize variation in the joint positions and input, without being adversely affected by balancing in non-vertical equilibrium positions.

3.4.2. *LQR Tuning by a Genetic Algorithm*

The GA was used to tune the four state-feedback gains of the LQR. Each controller gain was represented as a unique trait. Accordingly, there were four traits, represented with 15 genes per trait, for a total chromosome length of 60 genes. The allowable ranges for the feedback gains K_0, K_1, K_2, and K_3, were set at $[-9999.0,\ -40]$, $[-1000.0,\ -2.0]$, $[-2000.0,\ -5.0]$, and $[-300.0,\ -0.1]$, respectively. These ranges were chosen because they bound the gains determined by designing 20 LQR controllers – each designed with different weighting matrices. The population size for the GA was set to 20, and each individual in the initial population was 'seeded' with the parameters of one of the 20 previously designed LQRs.

This GA was run for 100 generations using the weighting vector

$$\mathbf{w} = [1,\ 1,\ 1,\ 100,\ 80,\ 5,\ 10,\ 8,\ 0.5]^T. \tag{30}$$

This choice of **w** places emphasis on minimizing the joint movements, with a lesser emphasis placed on the control input. The probabilities of mutation and

* The **if** condition in Equation (27) is used to ensure that the balancing fitness is not less than 0.001. We desire a fitness greater than zero for assigning a probability of mating to an individual in the GA.

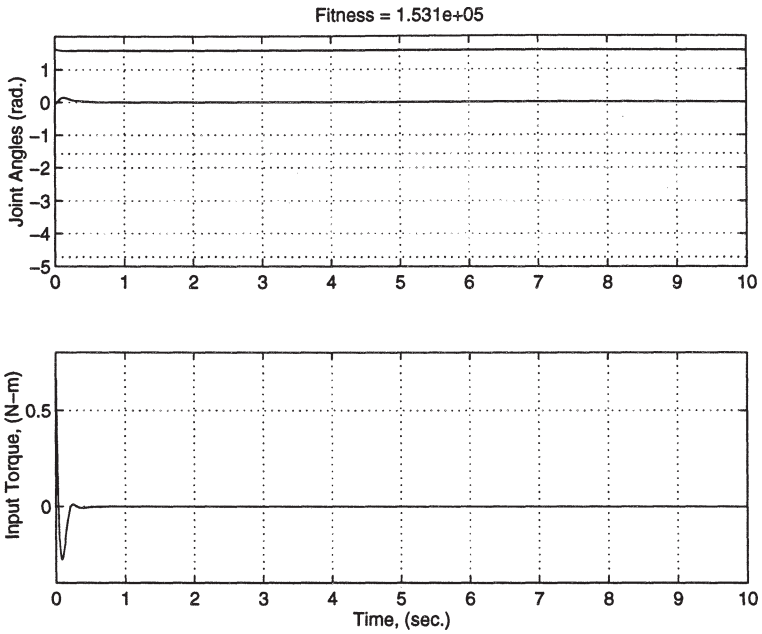

Figure 12. Simulation results for LQR tuned by a GA.

crossover were set to 0.05 and 0.5, respectively. It is important to note that we have seeded the GA with LQR gains, but that as the GA searches the gain space it is likely that it will not produce a set of LQR gains (i.e., ones that would minimize the performance index). An alternative approach to the GA tuning of the LQR would be to use a population of weighting matrices and at each step solve the Riccati equation. This would force the GA to search over the space of LQRs, but it would be much more computationally intensive and hence we have avoided this approach.

Simulation results for the linear controller tuned by the GA are shown in Figure 12. We have used the same initial conditions used throughout Section 3.2. We see that after tuning, the acrobot response is still quite smooth and uses very little input torque; however, the speed of the balancing response has been increased. This is not necessarily a beneficial characteristic though. Indeed it is difficult to say that the balancing performance has been improved. Simulations results using the re-tuned controller started from other initial conditions also yield similar results. The final gains determined by the GA are shown in Table II, along with the gains developed in Section 3.1 (the gains from Section 3.1 are denoted as 'Initial Value'). Notice how little each gain has been changed after tuning by the GA. Considering the large range for the gains of different LQR balancing controllers, the fact that the gains have remained almost fixed indicates that our original LQR was a very good balancing controller.

Table II. State-feedback gains before and after tuning by GA

Parameter	Initial value	Final value
K_0	-113.8908	-109.4879
K_1	-9.5070	-9.8571
K_2	-17.3951	-16.0464
K_3	-1.9405	-1.9738

3.4.3. *Direct Fuzzy Controller Tuning by a Genetic Algorithm*

The next balancing controller tuned by a GA was the direct fuzzy controller introduced in Section 3.2.1. The GA was used to tune six of the fuzzy controller parameters: the four input normalizing gains, the output gain, and the input–output mapping power (z). Each controller parameter was represented as a unique trait. Accordingly, there were 6 traits, represented with 15 genes per trait, making the total chromosome length for each individual 90 genes. The probabilities of mutation and crossover were again set to 0.05 and 0.5, respectively. The GA was run for 100 generations with a population size of 20. As a starting point, a single individual in the initial population was seeded with the parameters of the manually tuned fuzzy controller developed previously; all other individual's chromosomes were randomly determined such that each trait was within pre-specified allowable ranges for each controller parameter. Only a single individual was seeded with a working controller here because it is significantly more time consuming to determine working fuzzy controllers than LQRs. The allowable ranges for the fuzzy controller parameters, g_0, g_1, g_2, g_3, h, and z were set at, respectively: $[2.0,\ 8.0]$, $[0.1,\ 2.5]$, $[0.1,\ 2.0]$, $[0.01,\ 2.0]$, $[4.0,\ 99.0]$, and $[0.1,\ 1.5]$.

The weighting vector chosen first was the same vector used with the LQR:

$$\mathbf{w} = [1,\ 1,\ 1,\ 100,\ 80,\ 5,\ 10,\ 8,\ 0.5]^T. \tag{31}$$

The same initial condition as earlier was used to start each simulation. Recall that this initial condition resulted in a rather poor balancing performance for the manually tuned controller (see Figure 5). Note that using the fitness function developed in Section 3.4.1 and the weights above, the manually tuned fuzzy controller is given a fitness value of 231.2. The simulation results for the best direct fuzzy controller tuned by the GA are shown in Figure 13. The results indicate that the GA was able to greatly improve the performance of the direct fuzzy controller by tuning the six controller parameters previously indicated. We do still see a significant control input during the first few seconds of the simulation though.

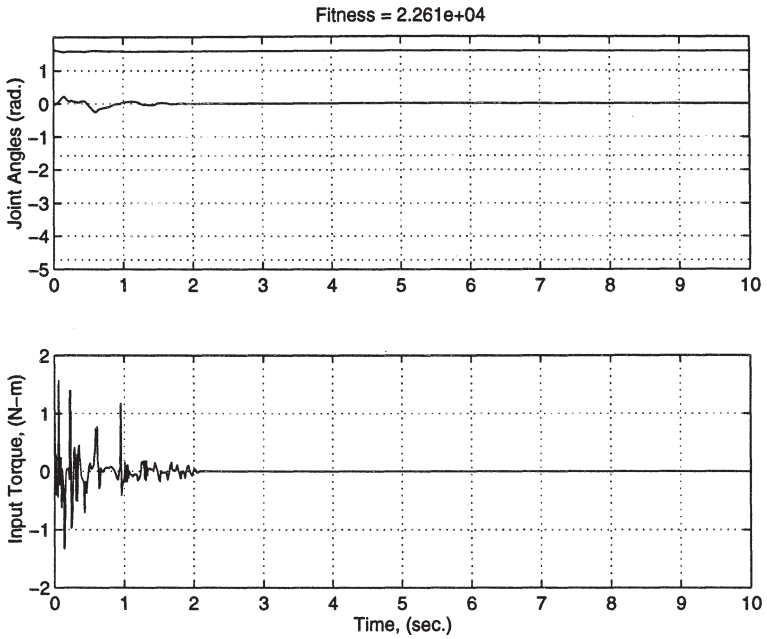

Figure 13. Simulation results for direct fuzzy controller tuned by a GA.

Next, in an attempt to reduce the required control input, the weighting vector in the balancing fitness function was changed to

$$\mathbf{w} = [1,\ 1,\ 1,\ 100,\ 80,\ 50,\ 100,\ 80,\ 50]^T. \tag{32}$$

This weighting vector greatly increases the weights on the transient characteristics of the controller (where the large torque inputs are found), and increases the weighting on the control input throughout the simulation. All other aspects of the GA were left unchanged from the previous run. Simulation results for the direct fuzzy controller tuned by the GA with this weighting vector are shown in Figure 14. While placing greater emphasis on the control input in the fitness function has significantly reduced the control input, it has not quite been reduced to the level of the LQR (compare with Figure 12).

A comparison between the manually tuned controller parameters and the controller parameters determined by the GAs with the different fitness functions is shown in Table III. Here we see that the GAs have changed the input normalizing gains of the fuzzy controller very little. The GA with the fitness function that stressed a reduction in control input achieved this objective by reducing the output gain and increasing the value of z. Both of these changes work together to reduce the overall controller gain. The other GA has changed all of the controller parameters only slightly. One might be tempted to say that the fuzzy controller with the reduced torque requirements is a more desirable controller. However, while this controller has met the objective of a reduced control input for this

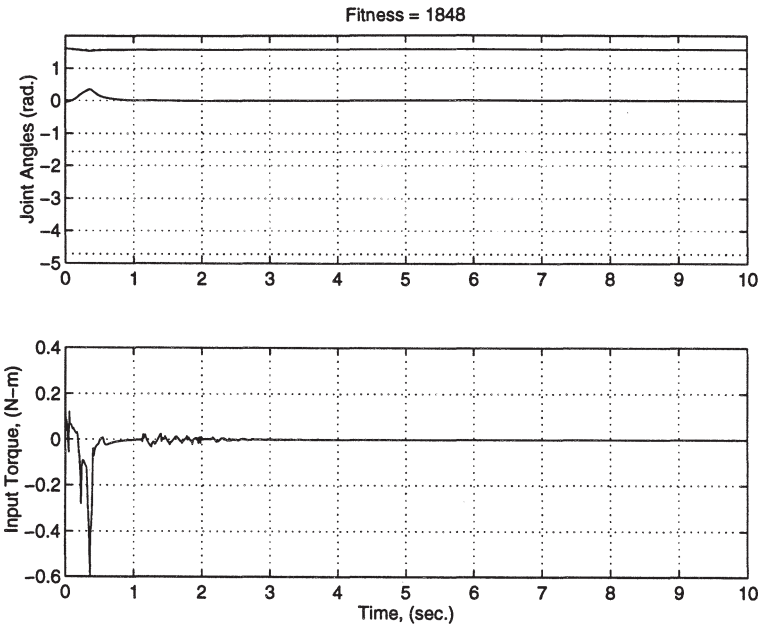

Figure 14. Simulation results for direct fuzzy controller tuned by a GA using a different weighting vector.

Table III. Direct fuzzy controller parameters before and after tuning by GA

Parameter	Initial value	Final value 1	Final value 2
g_0	5.5556	5.5572	5.4855
g_1	0.4708	0.4758	0.4096
g_2	0.8499	0.8089	0.8109
g_3	0.0951	0.0960	0.0956
h	55.9147	55.9143	25.6158
z	0.8000	0.9891	1.2016

initial condition, the re-tuned controller is now unable to balance the acrobot for some other initial conditions (though using the FMRLC, the re-tuned controller is still able to balance the acrobot for other initial conditions). This is not the case for the controller tuned with the other fitness function – it has resulted in improved controller performance for several initial conditions.

Note, that while the controller parameters are only shown to 4 decimal places in Table III, each controller parameter was actually represented with 15 genes on a chromosome (leaving 14 significant digits to represent the controller parameters plus one sign digit). This amount of precision *is* necessary. If such precision is not used to represent the controller parameters the results obtained for a single

independent simulation will not match the results obtained when run from within the GA. For example, if only 8 genes are used to represent each trait, a controller which produced a satisfactory balancing response in independent simulation may be unstable when run from within the GA. This is due to the loss of precision resulting from subsequently encoding and decoding the parameter values. Even using 15 genes to represent each trait results in different simulation results between the manually tuned controller simulated independently and the manually tuned controller simulated within the GA. The reverse situation, however, does not occur: the results obtained from within the GA can be exactly duplicated in independent simulation. This is because the actual parameters passed to the fuzzy controller are stored by the GA.

3.4.4. *FMRLC Tuning by a Genetic Algorithm*

A GA was next used to tune the same controller parameters of a direct fuzzy controller, except this time an FMRLC was added to the feedback loop. The weighting vector used was the same vector used in the first run above:

$$\mathbf{w} = [1,\ 1,\ 1,\ 100,\ 80,\ 5,\ 10,\ 8,\ 0.5]^T. \tag{33}$$

A single individual in the initial population was again initialized with the initial controller parameters of the manually tuned direct fuzzy controller. The FMRLC used was configured identically to the FMRLC developed in Section 3.2.2. All

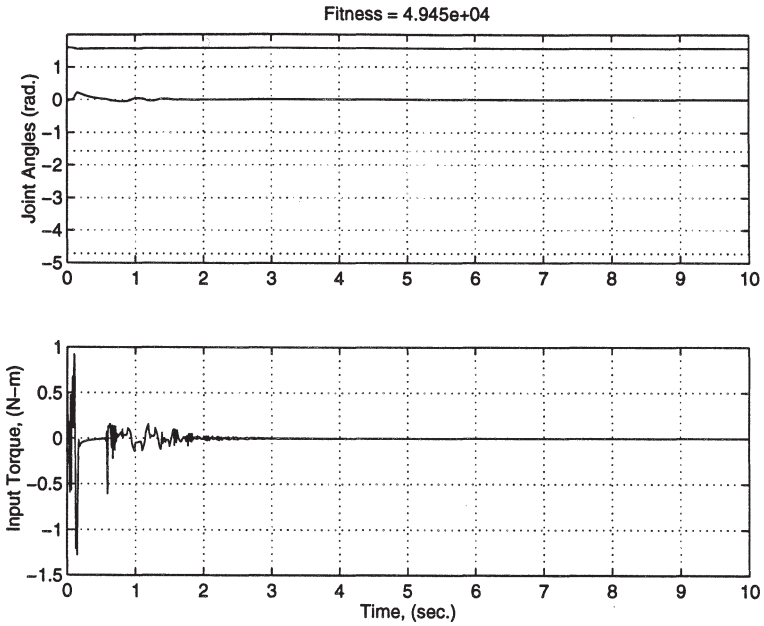

Figure 15. Simulation results for FMRLC tuned by a GA.

Table IV. Direct fuzzy controller parameters tuned by GA with FMRLC in feedback loop

Parameter	Initial value	Final value
g_0	5.5556	5.5155
g_1	0.4708	0.4852
g_2	0.8499	0.8120
g_3	0.0951	0.0942
h	55.9150	15.9027
z	0.8000	0.6620

of the FMRLC parameters remain fixed throughout the simulation. The GA was configured in exactly the same manner as it was for the direct fuzzy controller alone: probability of mutation is 0.05, probability of crossover is 0.5, population size is 20, and number of generations is 50. The initial conditions were also the same. Figure 15 shows the simulation of the best controller tuned by the GA.

The changes made to the controller parameters by the GA are shown in Table IV. Even using the fitness function which does not heavily emphasize a reduction in control input, the GA has tuned the FMRLC to a significantly reduced effective gain. This re-tuned controller is successful at balancing the acrobot from other initial conditions, but sometimes at a somewhat reduced performance. Thus, the FMRLC tuned by the GA does not always perform as well as the LQR (the GA tuning seems to have made the FMRLC more sensitive).

4. Swing-Up Controllers

Swing-up control for the acrobot involves commanding the acrobot to transition from the downward stable equilibrium point to an unstable inverted equilibrium position. Both joints must approach the inverted position with nearly zero velocity so that they may be ‘caught’ by a balancing controller. In this section three types of swing-up controllers will be developed. The three types of swing-up control to be studied include partial feedback linearization and subsequent PD control, classical state-feedback control, and direct fuzzy control. The parameters for each of the controllers mentioned will first be manually tuned; later, a GA will be used to re-tune the controller parameters. Simulation results and performance evaluations will be provided for each controller.

4.1. MANUALLY TUNED SWING-UP CONTROLLERS

This section will introduce three types of controllers to swing up the acrobot in simulation. All of the parameters for the controllers used in this section were manually tuned, some with greater difficulty than others.

4.1.1. *PD Control of the Linearized System*

Here we introduce an inner-loop partial feedback linearizing controller developed for the acrobot system. An outer-loop PD controller will then be used to swing up the linearized system. The first swing-up controller applied to the acrobot system utilizes partial feedback linearization and is the swing-up technique proposed by Spong in [3]. Feedback linearization is a nonlinear control technique that uses feedback to effectively cancel the nonlinearities of a system. As an example, consider the single input nonlinear system $\dot{\mathbf{x}} = \mathbf{f(x)} + \mathbf{g(x)}u$. The system is said to be feedback linearizeable if there exists a region U in $\mathbf{R}^n$, a diffeomorphism[⋆] T: $U \to \mathbf{R}^n$, and a nonlinear feedback $u = \alpha(\mathbf{x}) + \beta(\mathbf{x})v$, such that the transformed states $\mathbf{y} = \mathbf{T(x)}$ satisfy the linear equations $\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} + \mathbf{b}v$. Unfortunately, proving the existence of such a transformation and nonlinear feedback is often difficult. Another disadvantage of feedback linearization is that it relies on the exact cancellation of mathematical terms and therefore requires exact knowledge of the system dynamics and parameters. The main advantage of feedback linearization is, however, that it allows the control engineer to design a linear controller for a linear system – a well understood task.

In [3], Spong proposes a nonlinear feedback which results in partial linearization of the acrobot. The meaning of *partial* linearization will become apparent after the development to follow. Recall the dynamical equations for the acrobot are given by the two second order coupled differential equations

$$d_{11}\ddot{q}_1 + d_{12}\ddot{q}_2 + h_1 + \phi_1 = 0, \tag{34}$$

$$d_{12}\ddot{q}_1 + d_{22}\ddot{q}_2 + h_2 + \phi_2 = \tau. \tag{35}$$

By solving for $\ddot{q}_1$ in Equation (34) and substituting into Equation (35), we may rewrite Equation (35) as

$$\bar{d}_{22}\ddot{q}_2 + \bar{h}_2 + \bar{\phi}_2 = \tau, \tag{36}$$

where the terms $\bar{d}_{22}$, $\bar{h}_2$, and $\bar{\phi}_2$ are defined as[⋆⋆]

$$\begin{aligned} \bar{d}_{22} &= d_{22} - d_{12}d_{11}^{-1}d_{12}, \\ \bar{h}_2 &= h_2 - d_{12}d_{11}^{-1}h_1, \\ \bar{\phi}_2 &= \phi_2 - d_{12}d_{11}^{-1}\phi_1. \end{aligned} \tag{37}$$

It is easily shown that the nonlinear feedback

$$\tau = \bar{d}_{22}v_2 + \bar{h}_2 + \bar{\phi}_2 \tag{38}$$

[⋆] A **diffeomorphism** is a differentiable function whose inverse exists and is also differentiable [5].

[⋆⋆] Note that the parameter d_{11} is a scalar which is nonzero due to the positive definiteness of the acrobot's inertia matrix [3].

defines a feedback linearizing controller for Equation (36), with new input v_2.

By including the feedback linearizing control the system dynamics may be rewritten as

$$d_{11}\ddot{q}_1 + h_1 + \phi_1 = -d_{12}v_2, \tag{39}$$

$$\ddot{q}_2 = v_2. \tag{40}$$

Notice that Equation (40) is linear from input v_2 to output q_2. Thus, the nonlinear feedback law has linearized the system with respect to q_2 (partial linearization), and has also decoupled the motion of joint two from that of joint one. Equation (39) is still nonlinear, however, and the motion of joint one has not been decoupled from that of joint two; it is in fact the motion of joint two that excites joint one and enables the acrobot to swing up.

Indeed, only partial feedback linearization is possible for the acrobot system. Consider the acrobot dynamics written in the form

$$\dot{\mathbf{x}} = \mathbf{f(x)} + \mathbf{g(x)}u. \tag{41}$$

The overall system *cannot* be feedback linearizeable because the set*

$$\{\mathbf{g}, \mathbf{ad_f}(\mathbf{g}), \mathbf{ad_f^2}(\mathbf{g})\} \tag{42}$$

is not involutive [7, 5].

Utilizing the partial feedback linearization developed above, a proportional-derivative (PD) controller may be defined for q_2 as

$$v_2 = \ddot{q}_2^d + K_d(\dot{q}_2^d - \dot{q}_2) + K_p(q_2^d - q_2), \tag{43}$$

where q_2^d represents the desired position of the second link. In [3], Spong suggests the use of the arctangent function to generate the reference input q_2^d as a function of the velocity of joint one, particularly

$$q_2^d = \alpha \arctan(\dot{q}_1). \tag{44}$$

The arctangent function has the effect of pumping energy into joint one during each swing; however, any first and third quadrant function may be used [3]. The arctangent function has the desirable characteristic of straightening out the second joint when $\dot{q}_1$ equals zero at the peak of each swing, allowing a balancing controller to catch the system in the approximately inverted position.

This reference choice leads to an autonomous system, but requires a slightly altered control law if we assume that only q_1, q_2, $\dot{q}_1$, and $\dot{q}_2$ are measurable. A realizable control law given by

$$v_2 = K_p(q_2^d - q_2) - K_d\dot{q}_2 \tag{45}$$

* The term $\mathbf{ad_f^k}(\mathbf{g})$ denotes the iterative Lie Bracket $[\mathbf{f}, \mathbf{ad_f^{k-1}}(\mathbf{g})]$, where the Lie Bracket of $\mathbf{f}$ and $\mathbf{g}$ is defined as: $[\mathbf{f}, \mathbf{g}] = \frac{\partial \mathbf{g}}{\partial \mathbf{x}}\mathbf{f} - \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{g}$.

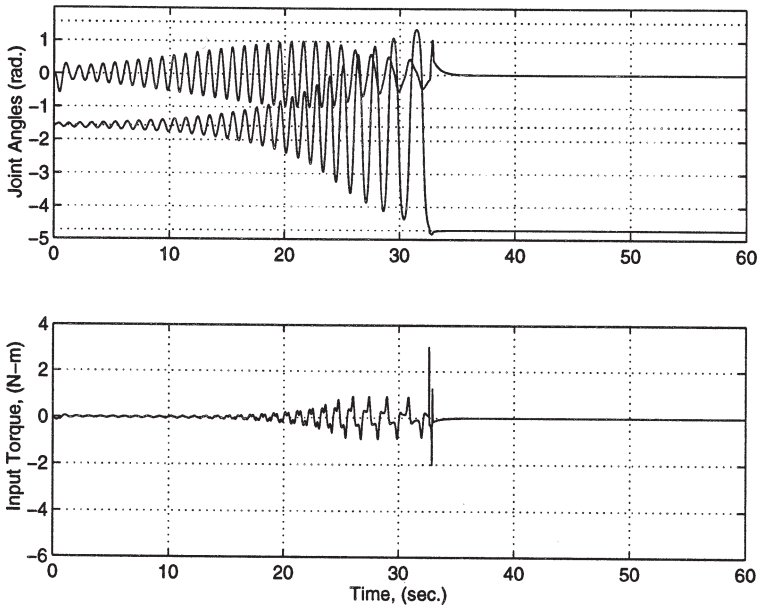

Figure 16. Simulation results for outer-loop PD controller with inner-loop partial feedback linearizing controller.

leads to the system input

$$\tau = \bar{d}_{22}[K_p(\alpha \arctan(\dot{q}_1) - q_2) - K_d\dot{q}_2] + \bar{h}_2 + \bar{\phi}_2. \tag{46}$$

The effect of using the realizable control law in Equation (45) instead of the control law in Equation (43) is that the motion of joint two is no longer exactly decoupled from that of joint one in a transformed coordinate system; however, this is not critical for the swing-up action [3].

While not immediately apparent, the applied torque given by Equation (46) is identically zero when the acrobot is started from the stable equilibrium point, $[q_1, q_2, \dot{q}_1, \dot{q}_2]^T = [-\pi/2, 0, 0, 0]^T$. Although any deviation from the equilibrium point will start the acrobot in motion, an exogenous input was added to the system for the first one half second so that the swing-up simulations could be started from the stable equilibrium point. The exogenous input is given by

$$\tau = \frac{\pi}{8}\sin(7.0\,t + \pi) \quad (t < 0.5\,\text{sec.}). \tag{47}$$

This function was found to smoothly start joints one and two in motion.

By tuning the gains K_p, K_d, and α, values were found which successfully swing up the acrobot. The LQR has been used to catch and balance the acrobot in the inverted position because it exhibited the best *overall* catching behavior for the nominal system. The balancing controller is engaged when the position of joint one is within β radians of $\pi/2$, or $|q_1 - \pi/2| < \beta$. A value of $\beta = 0.08$ radians was used in all of the swing-up simulations. Figure 16 shows the swing-

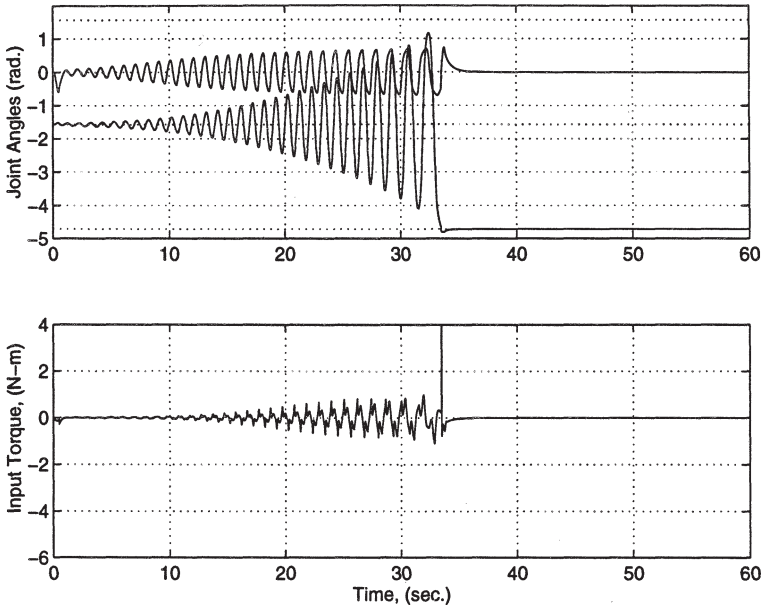

Figure 17. Simulation results for state-feedback PD controller.

up simulation results for the manually tuned PD controller with $K_p = 44.0$, $K_d = 2.1$, and $\alpha = 0.1$. Notice that this controller is able to swing up the acrobot in just over 30 seconds; however, the transition from swing-up to balancing controller is not particularly smooth.

4.1.2. *State Feedback Swing-Up*

While we have seen that the swing-up strategy utilizing partial feedback linearization and PD control is able to successfully swing up the acrobot, we had to assume complete knowledge of the system parameters to implement the control law. In reality, our knowledge of the system parameters is not exact; thus, we desire a swing-up control law that does not rely on such precise knowledge of the system parameters. In this section we will show how the acrobot may be successfully swung-up using a state-feedback approach. These controllers have been referred to as state-feedback controllers because only the system states are used as inputs to the controller; however they are not typical state-feedback controllers because the controller outputs are *not* a simple weighted sum of the inputs.

The first state-feedback swing-up controller tried was the same outer-loop PD controller used previously in Section 4.1.1; the dependence on exact parameter knowledge has been removed by eliminating the inner-loop control. This controller may be written as

$$\tau = K_p(q_2^d - q_2) - K_d\dot{q}_2 \tag{48}$$

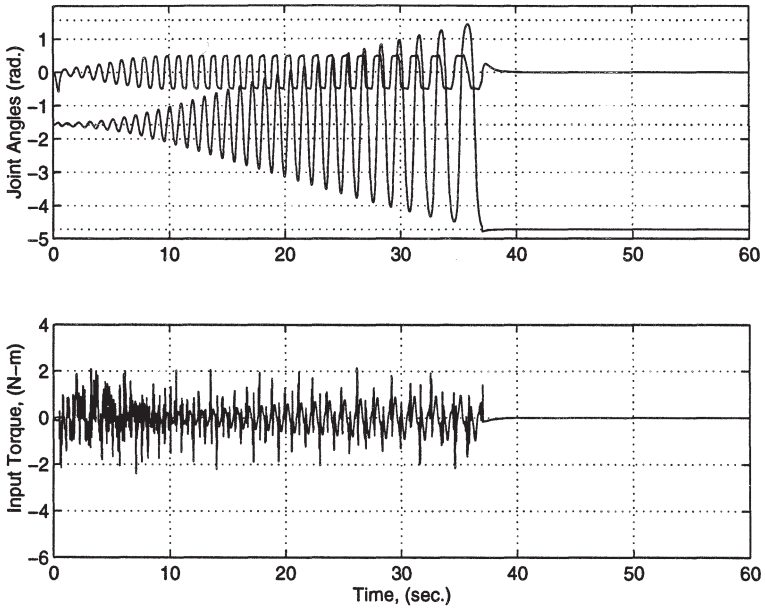

Figure 18. Simulation results for three-input fuzzy controller.

$$= K_p(\alpha \arctan(\dot{q}_1) - q_2) - K_d\dot{q}_2. \tag{49}$$

Gains which cause this controller to successfully swing up the acrobot were found with no greater difficulty than the gains for the system with inner-loop control. The swing-up response for gain values of $K_p = 10.1$, $K_d = 1.5$, and $\alpha = 0.505$ is shown in Figure 17. Here we see that with no more tuning effort than the previous PD controller with inner-loop feedback linearization, we have been able to swing up the acrobot in approximately the same time, with a comparable control input, and without using the plant parameters in the feedback law.

4.1.3. *Fuzzy Controller Swing-Up*

The first direct fuzzy swing-up controller tested was implemented in a manner similar to the classical swing-up controller given in Equation (48). A two-input fuzzy controller with inputs $g_0^{sw}(\alpha \arctan(\dot{q}_1) - q_2)$ and $g_1^{sw}\dot{q}_2$ was used as the swing-up controller, where g_0^{sw} and g_1^{sw} represent the fuzzy controller input normalizing gains. The single output from the fuzzy controller was multiplied by a constant gain h^{sw}. This controller was very difficult to tune manually, however, and no gains were found to result in a successful swing-up and balancing. The main problem associated with this controller was that joints one and two were found to swing in phase – precisely the phenomenon we had hoped to avoid with our choice of reference input. In the next section, we will show how a genetic GA is able to successfully tune the fuzzy controller in this form.

In an attempt to improve the phase between links one and two another input was added to the fuzzy controller: $g_2^{sw} q_2$. By using a gain $g_2^{sw} < 0$, the phase relationship can be greatly improved*. The idea for this input came from experience gained tuning an acrobot with different parameters. This acrobot had a tendency to swing link two beyond the physical π radian constraint. In this case, the addition of the term $g_2^{sw} q_2$ was found to not only reduce the magnitude of the second joint angle, but also to improve the phase between joints one and two. Using this term, the fuzzy controller was easily tuned to successfully swing up the acrobot. The simulation results for this fuzzy controller are shown in Figure 18. While the movement of links one and two appears to be similar to the other two swing-up controllers, the torque input is not nearly as smooth.

The fuzzy controller used has input normalizing gains of $g_0^{sw} = 1.00$, $g_1^{sw} = -0.10$, and $g_2^{sw} = -2.01$, and an output normalizing gain of 5.0. A value of 1.2 is used for α (note that α could have easily been incorporated into the gain g_0^{sw}). The fuzzy system has 15 triangular membership functions, with 50% overlap, evenly distributed over the three input universes of discourse. Nine triangular membership functions are distributed over the output universe of discourse, index adding was used as with the balancing controller, and the output membership function centers are mapped with a z value of 0.50.

4.2. SWING-UP CONTROLLER TUNING BY A GENETIC ALGORITHM

In this section the parameters of the manually tuned swing-up controllers introduced in Section 4.1 will be tuned by GAs. We will show how the GAs have improved the performance of each manually tuned swing-up controller. Additionally, the GAs have been able to tune successful swing-up controllers without seeding the initial population with the parameters of a working controller.

4.2.1. *The Swing-Up Fitness Function*

Determining a fitness function which quantifies the swing-up performance was a tedious albeit necessary process. The fitness function eventually determined places heavy emphasis on a reduced swing-up time and on the smoothness of the transition from swing-up to balancing controller. Similar to the fitness function developed for the balancing controller alone, the mean and sum of the squares of the two joint positions and input torque were acquired after the controller was switched from swing-up to balancing. The swing-up time and the maximum values attained by joints one and two were acquired during operation of the swing-up controller. Let t_{final} denote the simulation time length, and t_{switch} denote the time at which the controller is changed from a swing-up to a balancing controller. We now define the following variables used in determining the swing-

* For a conventional controller this would be no different than increasing K_p while decreasing α; however, the two are not equivalent for a fuzzy controller since it is nonlinear.

up fitness:

$$\text{stat}_0 = \begin{cases} 0 & \text{if (control}|_{(t=t_{\text{final}})} = \text{balance)} \\ 1 & \text{else,} \end{cases} \tag{50}$$

$$\text{stat}_1 = \begin{cases} 1 & \text{if } (|m_{q_1}| < 0.5) \\ 0 & \text{else,} \end{cases} \tag{51}$$

$$\text{stat}_2 = \frac{\text{stat}_0 \cdot 10.0}{\pi/2 - \max(q_1)}, \tag{52}$$

$$\text{stat}_3 = \frac{\text{stat}_1 \cdot 500 \cdot (60.0 - t_{\text{switch}})}{|m_{q_1}| + |m_{q_2}| + |m_u| + 10 \cdot s_{q_1} + 10 \cdot s_{q_2} + 5 \cdot s_\tau}, \tag{53}$$

where

$$m_{q_1} = \frac{\sum_{i[t=t_{\text{switch}}]}^{i[t=t_{\text{final}}]} (q_1[i] - \pi/2)}{i[t = t_{\text{final}}] - i[t = t_{\text{switch}}]}, \tag{54}$$

$$m_{q_2} = \frac{\sum_{i[t=t_{\text{switch}}]}^{i[t=t_{\text{final}}]} q_2}{i[t = t_{\text{final}}] - i[t = t_{\text{switch}}]}, \tag{55}$$

$$m_\tau = \frac{\sum_{i[t=t_{\text{switch}}]}^{i[t=t_{\text{final}}]} \tau}{i[t = t_{\text{final}}] - i[t = t_{\text{switch}}]}, \tag{56}$$

$$s_{q_1} = \frac{\sum_{i[t=t_{\text{switch}}]}^{i[t=t_{\text{final}}]} (q_1[i] - \frac{\pi}{2})^2}{i[t = t_{\text{final}}] - i[t = t_{\text{switch}}]}, \tag{57}$$

$$s_{q_2} = \frac{\sum_{i[t=t_{\text{switch}}]}^{i[t=t_{\text{final}}]} q_2^2}{i[t = t_{\text{final}}] - i[t = t_{\text{switch}}]}, \tag{58}$$

$$s_\tau = \frac{\sum_{i[t=t_{\text{switch}}]}^{i[t=t_{\text{final}}]} \tau^2}{i[t = t_{\text{final}}] - i[t = t_{\text{switch}}]}. \tag{59}$$

The variables m_x and s_x in Equations (54)–(59) denote the normalized *balancing* mean of variable x and the normalized *balancing* sum of the squares of variable x, respectively. As in Section 3, $y[i]$ denotes the ith sample for variable y and $i[t = x]$ denotes the ith data sample taken at time $t = x$ seconds. Using these variables, the swing-up fitness function may be written as

$$f_s = \begin{cases} \text{stat}_2 + \text{stat}_3 & \text{if } \begin{pmatrix} (\text{stat}_2 + \text{stat}_3 > 0.001) \\ \text{and} \\ (\max(q_2) < \frac{\pi}{2} - 0.05) \end{pmatrix} \\ 0.001 & \text{else.} \end{cases} \tag{60}$$

The variable stat_0 is used to record whether or not the balancing controller was engaged by completion of the simulation*. The variable stat_1 attempts to quantify if the balancing controller was stable**. The variables stat_2 and stat_3 are the important variables in calculating the fitness function. If the balancing controller was not engaged at the end of the simulation (q_1 was never within β radians of $\pi/2$), stat_2 records the swing-up fitness ($\text{stat}_3 = 0$); as joint one swings closer to $\pi/2$ the value of stat_2 increases. If the balancing controller has been engaged ($\text{stat}_2 = 0$), and is stable, stat_3 records the swing-up fitness; stat_3 is highly dependent on the swing-up time, t_{switch}, due to the multiplicative term $(60 - t_{\text{switch}})$ in the numerator of stat_3. It is also dependent on the performance of the balancing controller (which is itself heavily dependent on initial conditions and hence the swing-up control) due to the denominator of stat_3. The gains used to define stat_2 and stat_3 ensure that any swing-up controller which results in balancing and catching of the acrobot in the inverted position receives a higher fitness than any swing-up controller which never engages the balancing controller. A swing-up controller which leads to an unstable balancing control receives the lowest possible fitness. The conditions in Equation (60) perform two functions: (i) they ensure that the minimum fitness is 0.001, and (ii) they ensure that individuals which swing joint two close to the $\pi/2$ limitation are not found to be fit.

This fitness function has the important property of increasing incrementally for simulations which move the acrobot closer to a successful swing-up; this was not the case for the balancing fitness function, which has the same low fitness value for all unsuccessful balancing controllers. The swing-up fitness function developed here will be used unchanged by each of the GAs used to tune the controllers in the upcoming sections.

4.2.2. *Outer-Loop PD Controller Tuning by a Genetic Algorithm*

The first use of a GA was in the tuning of the two PD gains, K_p and K_d, and the gain α for the outer-loop PD controller used in conjunction with the inner-loop partial feedback linearizing controller. The GA was configured with probabilities of mutation and crossover of 0.1 and 0.5, respectively; it was run for 100 generations with 20 individuals in the population. Each trait on the chromosome was represented with 15 genes, for a total chromosome length of 45 genes. The allowable ranges for K_p, K_d, and α were $[10.0,\ 90.0]$, $[0.1,\ 10.0]$, and $[0.1,\ 3.0]$, respectively. A single individual in the initial population was seeded with the manually tuned PD controller parameters used in Section 4.1.1 for the the first run of the GA. Using the fitness function developed in Section 4.2.1, the manually tuned swing-up controller (and initial population seed) was found to

* The current controller is stored in the variable *control*; this variable may take on a value of *swing-up* or *balance*.

** Note, that if the balancing controller was not engaged, $m_{q_1} = 0$.

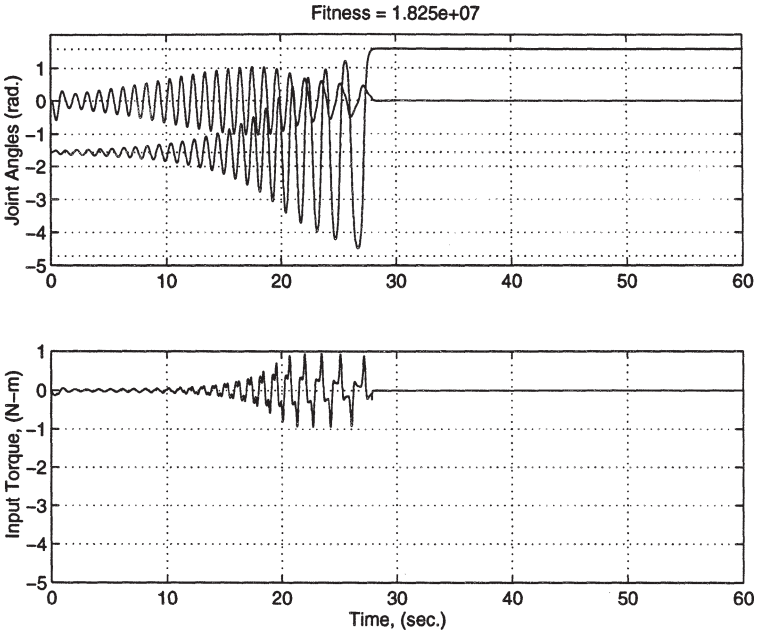

Figure 19. Simulation results for partial feedback linearization with PD gains tuned by a GA with a seeded population.

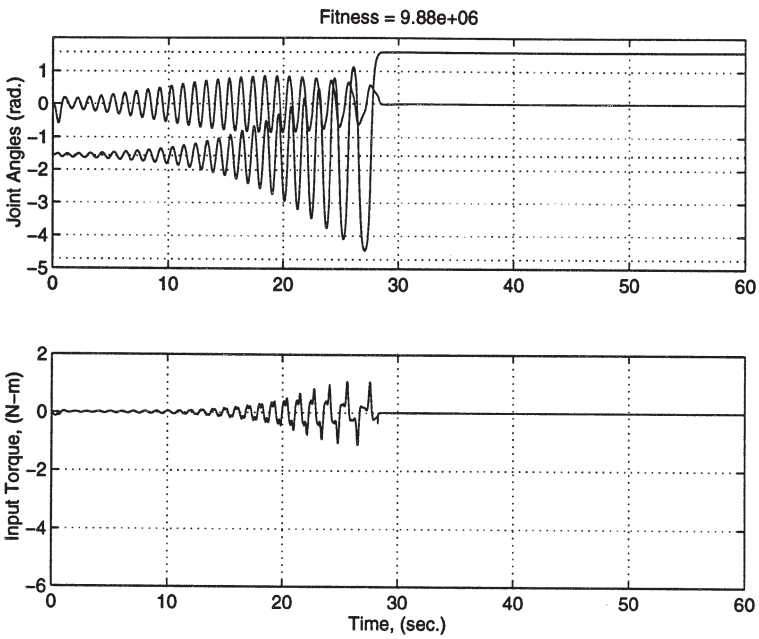

Figure 20. Simulation results for partial feedback linearization with PD gains tuned by a GA without a seeded population.

Table V. PD controller parameters and fitness tuned by a GA with and without a seeded population

Parameter	Manually tuned	Seeded population	Unseeded population
K_p	44.0000	46.0042	57.6571
K_d	2.1000	2.0500	4.0381
α	0.2200	0.2081	0.2854
f_s	7.6912×10^4	1.8250×10^7	9.8800×10^6

have a fitness of $f_s = 7.691 \times 10^4$. The simulation results for the PD controller tuned by the GA are shown in Figure 19. These results reveal that the GA has improved the performance of the swing-up controller by reducing the swing-up time to under 30 seconds and significantly smoothing out the transition from swing-up to balancing controller.

Next, a GA was used to tune the same controller without seeding the initial population; all other parameters remained unchanged for this run of the GA. The simulation results for the unseeded case are shown in Figure 20. Here we see that the simulation results obtained by the controller tuned by the unseeded GA are virtually identical to the results tuned by the seeded algorithm. The parameter values for the manually tuned PD controller and the parameters of the two controllers tuned by GAs are given in Table V. Notice how the parameter values tuned by the seeded GA have remained fairly close to the manually tuned values, while the unseeded GA has tuned the parameters quite differently. Indeed, for swing-up there are many parameter value combinations that produce similar simulation results.

4.2.3. *State-Feedback Controller Tuning by a Genetic Algorithm*

A GA was next used to tune the manually tuned controller given in Equation (48). The GA was configured to tune the gains K_p, K_d, and α. The probabilities of mutation and crossover were again chosen to be 0.1 and 0.5, respectively. The population size and number of generations was set at 20 and 50, respectively. Each trait on the chromosome was represented with 15 genes, for a total chromosome length of 60 genes. The allowable ranges for K_p, K_d, and α were $[0.1,\ 10]$, $[-5.0,\ 0.0]$, and $[0.1,\ 3.0]$, respectively. An individual in the initial population was once again seeded with the parameters of the manually tuned state-feedback controller; this controller was determined to have a fitness value of $f_s = 8.713 \times 10^4$. The simulation results for the re-tuned controller are shown in Figure 21. The GA has been able to greatly improve the swing-up performance by reducing the swing-up time and smoothing the transition between controllers. While the transition from swing-up to balancing controllers is not as smooth as

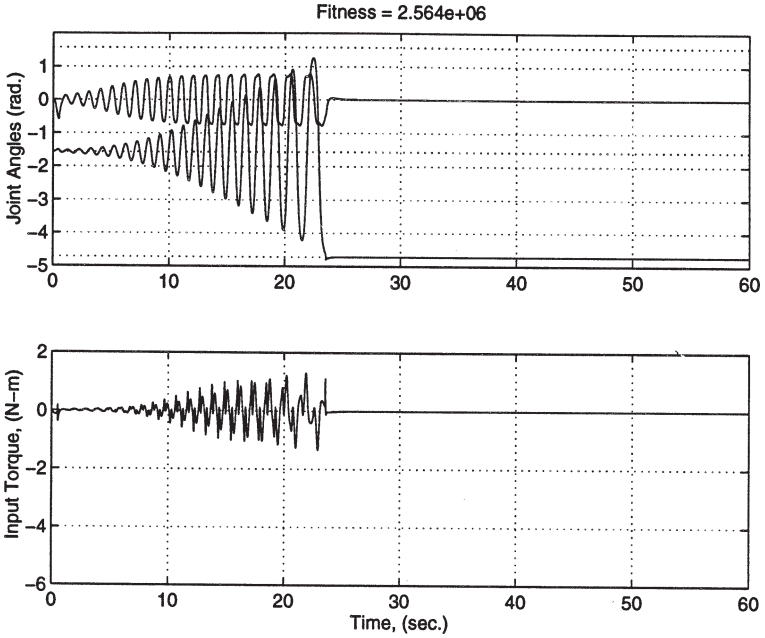

Figure 21. Simulation results for state-feedback controller with gains tuned by a GA with a seeded population.

Table VI. State-feedback controller parameters and fitness tuned manually and by a GA with and without a seeded population

Parameter	Manually tuned	Seeded population	Unseeded population
K_p	10.1000	10.0986	27.3830
K_d	1.5000	0.6548	3.1322
α	0.5050	0.5447	0.3227
f_s	8.713×10^4	2.564×10^6	1.079×10^6

for the previous PD controller, the swing-up time has been further reduced. It also appears that slightly more input torque has been used with this controller.

4.2.4. *Fuzzy Controller Tuning by a Genetic Algorithm*

A GA was next applied to tuning the fuzzy swing-up controllers introduced in Section 4.1.3. The first fuzzy controller tuned by a GA was the three-input fuzzy controller with inputs $g_0^{sw}(\alpha \arctan(\dot{q}_1) - q_2)$, $g_1^{sw}\dot{q}_2$, and $g_2^{sw}q_2$. The GA was configured to tune the three input normalizing gains: g_0^{sw}, g_1^{sw}, and g_2^{sw}; the output normalizing gain h^{sw}; the I/O mapping parameter z; and the arctangent

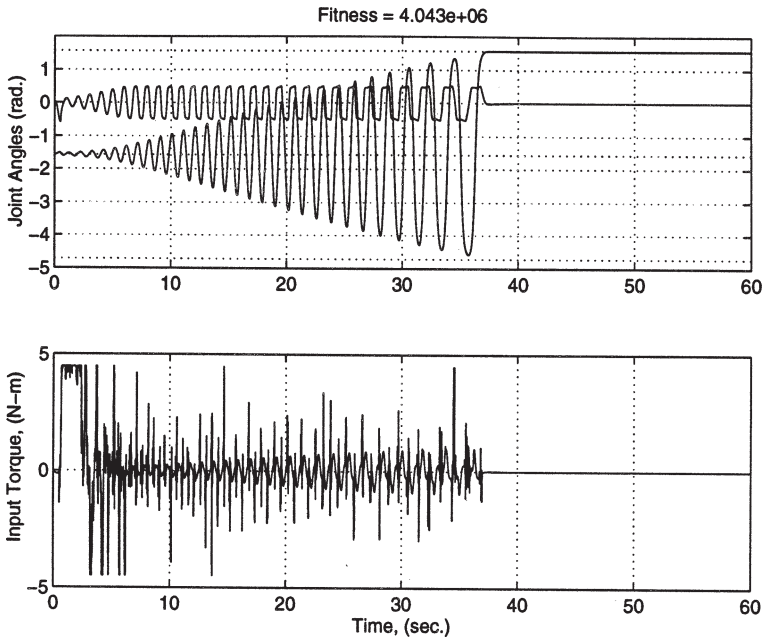

Figure 22. Simulation results for fuzzy controller with gains tuned by a GA with a seeded population.

gain α. The allowable ranges for g_0^{sw}, g_1^{sw}, g_2^{sw}, h^{sw}, z, and α were, respectively: $[0.1,\ 5.0]$, $[-5.0,\ 0.0]$, $[-6.0,\ 0.0]$, $[1.0,\ 15.0]$, $[0.1,\ 3.0]$, and $[0.1,\ 3.0]$. The probabilities of mutation and crossover were set to 0.1 and 0.5, respectively; 15 genes were used to represent each trait on the chromosome, making the total chromosome length 90 genes. The GA was run for 50 generations with 20 individuals per generation. For the first run, an individual in the initial population was seeded with the manually tuned fuzzy controller given in Section 4.1.3. The manually tuned fuzzy swing-up controller was determined to have a fitness of $f_s = 2.739 \times 10^5$. The simulation results for the fuzzy controller tuned by the GA are shown in Figure 22. This re-tuned controller has demonstrated the least improvement in performance. While the fitness has increased, it is seemingly a result only of smoothing the switching transition, as the swing-up time has remained approximately the same.

The same GA was then run again with the same configuration, except the initial population was not seeded with a tuned controller. The results for this case are shown in Figure 23. The parameter values of the manually tuned fuzzy controller and those determined by the seeded and unseeded GAs are shown in Table VII. We see that the unseeded case has resulted in a much faster swing-up time than the seeded case – the fastest swing-up yet. The fitness value for this swing-up simulation is not significantly higher than the simulation for the

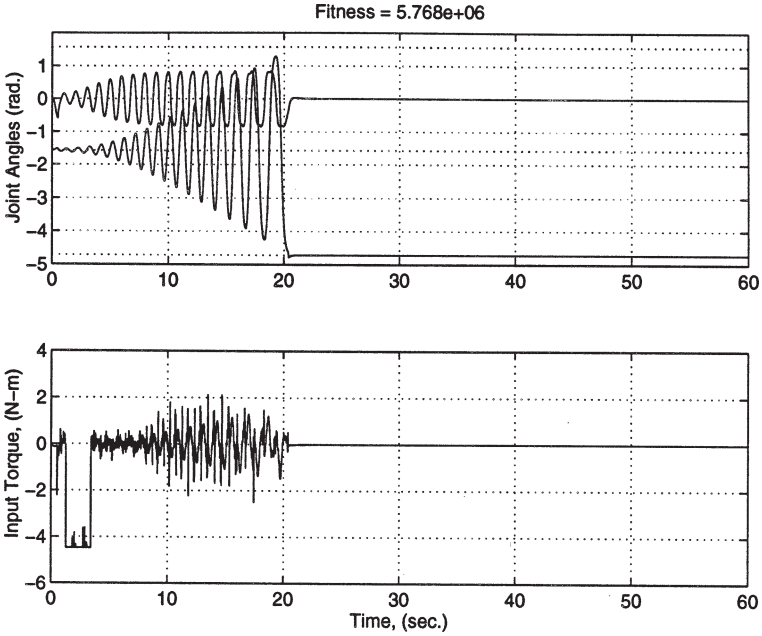

Figure 23. Simulation results for fuzzy controller tuned by a GA without a seeded initial population.

Table VII. Fuzzy controller parameters and fitness tuned by a GA with and without a seeded population

Parameter	Manually tuned	Seeded population	Unseeded population
g_0^{sw}	1.0000	1.0000	4.6983
g_1^{sw}	−0.1000	−0.1001	−0.1267
g_2^{sw}	−2.0100	−2.0094	−1.0323
h^{sw}	5.0000	14.7696	8.9313
z	0.5000	1.5519	1.9211
α	1.2000	1.4145	0.7162
f_s	2.739×10^5	4.043×10^6	5.768×10^6

controller tuned by the seeded GA because the transition between swing-up and balancing controllers is not as smooth as in the seeded case.

5. Concluding Remarks

Using the nonlinear model of the acrobot specified in Section 2, we were able to design a successful LQR balancing controller (in simulation) by linearizing the system dynamics about the inverted position. A successful direct fuzzy balancing

controller was then designed by emulating the action of a LQR. This was the only design method which leads to a working fuzzy balancing controller. Even after having gained considerable experience with the acrobot system dynamics, we were unable to transfer this knowledge to a working fuzzy balancing controller in the usual way that fuzzy controllers are designed. While the direct fuzzy controller designed for the acrobot was able to balance the acrobot in the inverted position in simulation, the state trajectories were not nearly as smooth as with the LQR. The fuzzy controller also commanded a larger, more oscillatory torque input. To improve the balancing performance of the direct fuzzy controller we then moved to an adaptive fuzzy controller: the FMRLC. The FMRLC was found to perform significantly better than the direct fuzzy controller, though still not as well as the LQR (all things considered). The three balancing controllers were also tested in simulation with different disturbances. With the exception of a random disturbance, the FMRLC was able to completely remove the effects of the disturbance, and therefore performed much better than either the LQR or direct fuzzy controller.

GAs were used to tune the parameter values of each type of balancing controller. We saw that a GA was able to improve the performance of each controller type for a specific initial condition (though it is difficult to determine if the performance of the LQR was really improved, since the responses before and after tuning were quite similar). While tuning by a GA did not adversely affect the performance of the LQR for initial conditions other than the initial condition used to tune the parameters, the direct fuzzy controller was found to be unstable for several other initial conditions after tuning. Indeed, the direct fuzzy controller was much more sensitive to different initial conditions than the LQR. Using the FMRLC, however, we found that we were able to eliminate this dependence on initial conditions observed with the direct fuzzy controller. In terms of overall robustness to changes in initial conditions, we observed that the non-linear controllers performed better before tuning by a GA than after – which makes sense since the GA tuned the controller for only a specific initial condition.

In Section 4 we developed three types of swing-up controllers for the acrobot. While the PD controller with partial feedback linearizing inner-loop controller was originally proposed by Spong, the state-feedback controller and fuzzy controller introduced have not previously been applied to the acrobot. Unlike the controller proposed by Spong, the latter two swing-up controllers do not require knowledge of the system parameters for construction of the controller. These controllers were found to perform as well as the PD controller with inner-loop linearizing control in simulation. We found that there were many combinations of controller parameter values which can be used to swing up the acrobot. For this reason, the swing-up controllers were well-suited to tuning by a GA. Additionally, we were able to use a fitness function which increased incrementally as we approached a working controller; this was not the case with the fitness function used in tuning the balancing controllers. Indeed, a GA was able to tune

successful swing-up controllers without seeding the initial population with the parameters of a working controller. Moreover, the controllers tuned by GAs with unseeded initial populations were found to perform as well as those tuned by GAs with seeded populations.

Acknowledgment

The authors would like to thank Stephen Yurkovich for several helpful comments on this work. We would also like to thank Will Lennon for the use of his GA C code.

References

1. Antsaklis, P. and Passino, K.: *An Introduction to Intelligent and Autonomous Control*, Kluwer Academic Publishers, Norwell, MA, 1993.
2. Gupta, M. and Sinha, M.: *Intelligent Control: Theory and Practice*, IEEE Press, Piscataway, NJ, 1995.
3. Spong, M.: Swing up control of the acrobot, in *IEEE Conference on Robotics and Automation*, San Diego, CA, 1994, pp. 2356–2361.
4. Spong, M.: The swing up control problem for the acrobot, *IEEE Control Systems Magazine*, 1995.
5. Spong, M. and Vidyasagar, M.: *Robot Dynamics and Control*, John Wiley and Sons, New York, 1989.
6. Bortoff, S. and Spong, M.: Pseudolinearization of the acrobot using spline functions, in *Proceedings of the 31st Conference on Decision and Control*, Tucson, AZ, 1992, pp. 593–598.
7. Hauser, J. and Murray, R.: Nonlinear controllers for non-integrable systems: The acrobot example, in *Proc. American Control Conference*, 1990, pp. 669–671.
8. Passino, K. and Yurkovich, S.: Fuzzy control, in W. Levine (ed.), *Handbook on Control*, CRC Press, Boca Raton, 1996.
9. Driankov, D., Hellendoorn, J. and Reinfrank, M.: *An Introduction to Fuzzy Control*, Springer-Verlag, New York, 1993.
10. Wang, L.: *Adaptive Fuzzy Systems and Control: Design and Stability Analysis*, Prentice-Hall, NJ, 1994.
11. Layne, J. and Passino, K.: Fuzzy model reference learning control for cargo ship steering, *IEEE Control Systems Magazine* **13**(6) (1993), 23–34.
12. Kwong, W. and Passino, K.: Dynamically focused learning control, *IEEE Transactions on Systems, Man, and Cybernetics*, **26**(1) (1996), 53–74.
13. Goldberg, D.: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
14. Michalewicz, Z.: *Genetic Algorithms + Data Structure = Evolution Programs*, Springer-Verlag, Berlin, Heidelberg, 1992.
15. Lee, M. A. and Takagi, H.: Integrating design stages of fuzzy systems using genetic algorithms, in *Second IEEE International Conference on Fuzzy Systems*, San Francisco, CA, 1993, pp. 612–617.
16. Varšek, A., Urbančič, T. and Filipič, B.: Genetic algorithms in controller design and tuning, *IEEE Transactions on Systems, Man and Cybernetics* **23**(5) (1993), 1330–1339.
17. Porter, B. and Borairi, M.: Genetic design of linear multivariable feedback control systems using eigenstructure assignment, *International Journal of Systems Science* **23**(8) (1992), 1387–1390.
18. Michelewicz, Z.: Genetic algorithms and optimal control problems, in *Proceedings of the 29th Conference on Decision and Control* (Honolulu, Hawaii), 1990, pp. 1664–1666.

19. Ishibuchi, H., Nozaki, K. and Yamamoto, N.: Selecting fuzzy rules by genetic algorithm for classification problems, in *Second IEEE International Conference on Fuzzy Systems*, San Francisco, CA, 1993, pp. 1119–1124.
20. Katai, O.: Constraint-oriented fuzzy control schemes for cart-pole systems by goal decoupling and genetic algorithms, in A. Kandel and G. Langholz (eds), *Fuzzy Control Systems*, CRC Press, Boca Raton, 1994, pp. 181–195.
21. Karr, C. and Gentry, E.: Fuzzy control of ph using genetic algorithms, *IEEE Transactions on Fuzzy Systems* **1**(1) (1993), 46–53.
22. Nomura, H., Hayashi, I. and Wakami, N.: A self-tuning method of fuzzy reasoning by genetic algorithm, in A. Kandel and G. Langholz (eds), *Fuzzy Control Systems*, CRC Press, Boca Raton, 1994, pp. 338–354.
23. Kwong, W., Passino, K., Laukonen, E. and Yurkovich, S.: Expert supervision of fuzzy learning systems for fault tolerant aircraft control, *Proceedings of the IEEE* **83**(3) (1995), 466–483.
24. Widjaja, M.: Intelligent control for swing up and balancing of an inverted pendulum, Master's thesis, The Ohio State University, 1994.
25. Layne, J. and Passino, K.: Fuzzy model reference learning control, in *Proc. 1st IEEE Conf. on Control Applications*, Dayton, OH, 1992, pp. 686–691.