A CASE STUDY IN APPROXIMATE LINEARIZATION: THE ACROBOT EXAMPLE

by

Richard M. Murray and John Hauser

Memorandum No. UCB/ERL M91/46

29 April 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A CASE STUDY IN APPROXIMATE LINEARIZATION: THE ACROBOT EXAMPLE

by

Richard M. Murray and John Hauser

Memorandum No. UCB/ERL M91/46

29 April 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A CASE STUDY IN APPROXIMATE LINEARIZATION: THE ACROBOT EXAMPLE

by

Richard M. Murray and John Hauser

Memorandum No. UCB/ERL M91/46

29 April 1991

A Case Study in Approximate Linearization: The Acrobot Example

Richard M. Murray*
Electronics Research Laboratory
University of California
Berkeley, CA 94720
murray@united.berkeley.edu

John Hauser†
Department of EE-Systems
University of Southern California
Los Angeles, CA 90089–0781
hauser@nyquist.usc.edu

29 April 1991

Abstract

The acrobot is a simple mechanical system patterned after a gymnast performing on a single parallel bar. By swinging her legs, a gymnast is able to bring herself into an inverted position with her center of mass above the part and is able to perform manuevers about this configuration. This report studies the use of nonlinear control techniques for designing a controller to operate in a neighborhood of the manifold of inverted equilibrium points. The techniques described here are of particular interest because the dynamic model of the acrobot violates many of the necessary conditions required to apply current methods in linear and nonlinear control theory.

The approach used in this report is to approximate the system in such a way that the behavior of the system about the manifold of equilibrium points is correctly captured. In particular, we construct an approximating system which agrees with the linearization of the original system on the equilibrium manifold and is full state linearizable. For this class of approximations, controllers can be constructed using recent techniques from differential geometric control theory. We show that application of control laws derived in this manner results in approximate trajectory tracking for the system under certain restrictions on the class of desired trajectories. Simulation results based on a simplified model of the acrobot are included.

*Research supported in part by an IBM Manufacturing fellowship and the National Science Foundation, under grant IRI-90-14490.

†Fred O'Green Assistant Professor of Engineering

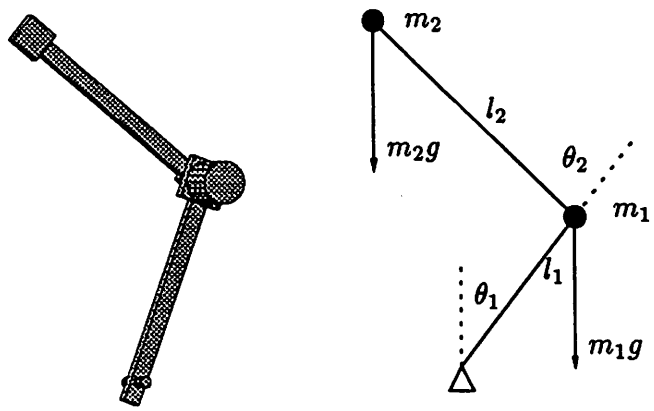

Figure 1: Acrobot: an acrobatic robot. Patterned after a gymnast on a parallel bar, the acrobot is only actuated at the middle (hip) joint; the first joint, corresponding to the gymnast's hands on the bar, is free to spin about its axis.

1 Introduction

Recent developments in the theory of geometric nonlinear control provide powerful methods for controller design for a large class of nonlinear systems. Many systems, however, do not satisfy the restrictive conditions necessary for either full state linearization [7, 5] or input-output linearization with internal stability [2]. In this paper, we present an approach to controller design based on finding a linearizable nonlinear system that well approximates the true system over a desirable region. We outline an *engineering* procedure for constructing the approximating nonlinear system given the true system. We demonstrate this approach by designing a nonlinear controller for a simple mechanical system patterned after a gymnast performing on a single parallel bar.

There has been considerable work in the area of system approximation including Jacobian linearization, pseudo-linearization [10, 12], approximation with a nonlinear system [8], and extended linearization [1]. Much of the work on system approximation has been directed toward analysis and the development of conditions that must be satisfied by the approximate systems rather than on the explicit construction of such approximations. Notable exceptions include the standard Jacobian approximation and the recent work of Krener using polynomial system approximations [9]. Wang and Rugh [12] also provide an approach for constructing configuration sched-

uled linear transformations to pseudo-linearize the system (note that this approach provides a *family* of approximations rather that a *single* system approximation). Rather than using polynomial systems or families of linear systems to approximate the given system, we approximate the given nonlinear system with a single nonlinear system that is full state linearizable.

We use as a guiding example the problem of controlling the *acrobot* (for acrobatic-robot) shown in Figure 1. The acrobot is a highly simplified model of a human gymnast performing on a single parallel bar. By swinging her legs (a rotation at the hip) the gymnast is able to bring herself into a completely inverted position with her straightened legs pointing upwards and her center of mass above the bar. The acrobot consists of a simple two link manipulator operating in a vertical plane. The first joint (corresponding to the gymnast's hand sliding freely on the bar) is free to rotate. A motor is mounted at the second joint (between the links) to provide a torque input to the system (corresponding to the gymnast's ability to generate torques at the hip). A life size acrobot is currently being instrumented for experimentation at U.C. Berkeley.

The eventual goal in controlling this system is to precisely execute realistic gymnastic routines. Our modest initial goal is to understand and design controllers capable of system control in a neighborhood of the manifold of inverted equilibrium positions. That is, we would like to have the acrobot follow a smooth trajectory while inverted, such as that shown in Figure 2.

This report presents a detailed study of the stabilization and tracking for the acrobot. We begin with a complete, mathematical description of the system in Section 2. The application of standard control techniques to the acrobot is studied in Section 3. Section 4 briefly introduces the theory of approximate linearization and develops a family of nonlinear controllers using this theory. A comparison of these controllers against a standard linear controller is given in Section 5. Finally, we discuss more general nonlinear control problems and how our results for the acrobot can be applied to them.

The application of the methods presented here require substantial algebraic computation. We have used Mathematica [13] to perform much of our computation for us. We list in the body of this paper the specific Mathematica files which were used to obtain or check indicated results. The listings for these files can be found in the appendix.

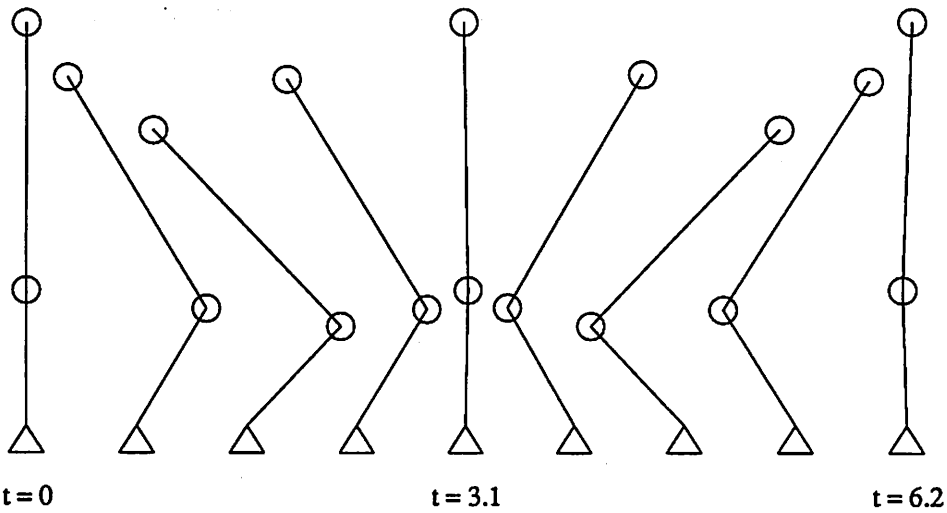

Figure 2: Motion of the acrobot along the manifold of inverted equilibrium positions.

2 System description

Considered as a mechanical system, the acrobot has unforced dynamics identical to those of a two link robot. Using a Lagrangian analysis (see for example [11]), the dynamics of the acrobot can be written as

$$M(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + G(\theta) = \begin{pmatrix} 0 \\ \tau \end{pmatrix}$$

where $\theta = (\theta_1, \theta_2)$ is the vector of relative joint angles as shown in Figure 1, M is the (uniformly positive definite) inertia tensor, C contains the Coriolis and centrifugal forces, G contains the effects of gravity, and τ is the torque applied between the first and second links. Using point mass approximations, a simple analysis yields (`acrobot.m`)

$$M(\theta) = \begin{bmatrix} a + b + 2c\cos\theta_2 & b + c\cos\theta_2 \\ b + c\cos\theta_2 & b \end{bmatrix} \tag{1}$$

$$C(\theta,\dot{\theta}) = \begin{bmatrix} -c\sin\theta_2\,\dot{\theta}_2 & -c\sin\theta_2(\dot{\theta}_1 + \dot{\theta}_2) \\ c\sin\theta_2\,\dot{\theta}_1 & 0 \end{bmatrix} \tag{2}$$

$$G(\theta) = \begin{bmatrix} -d\sin\theta_1 - e\sin(\theta_1 + \theta_2) \\ e\sin(\theta_1 + \theta_2) \end{bmatrix} \tag{3}$$

where

$$\begin{aligned} a &= m_1 l_1^2 + m_2 l_1^2 & \qquad d &= g m_1 l_1 + g m_2 l_1 \\ b &= m_2 l_2^2 & e &= g m_2 l_2 \\ c &= m_2 l_1 l_2 \end{aligned}$$

Due to the presence of rotary joints, these dynamics are highly nonlinear and contain many important trigonometric terms. Defining

$$x := \begin{pmatrix} \theta \\ \dot{\theta} \end{pmatrix}$$

we can write the system as a standard nonlinear system, affine in the control $u := \tau$,

$$\dot{x} = f(x) + g(x)u$$

where the system vector fields, f and g, are given by

$$f(x) := \begin{pmatrix} \dot{\theta} \\ -M^{-1}(C\dot{\theta} + G) \end{pmatrix} \text{ and } g(x) := \begin{pmatrix} 0 \\ M^{-1}\binom{0}{1} \end{pmatrix} \tag{4}$$

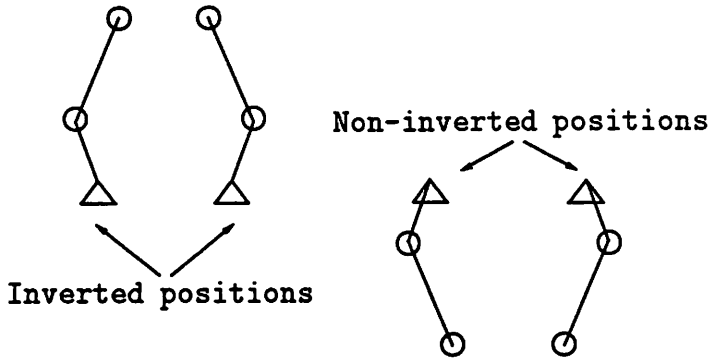

Figure 3: Equilibrium points for $\theta_2 = \alpha$. In general the inverted equilibrium points are in a separate component of the equilibrium set from the non-inverted ones.

Since the system has a single input, we can find a one-dimensional set of equilibrium points (e.g., inverted positions) that the system can achieve. This set consists of all states where $f(x_0) + g(x_0)u_0 = 0$ for some input u_0. In particular, this is only true if

$$\begin{aligned} \dot{\theta} &= 0 \\ G(x_0) &= \binom{0}{1} u_0 \end{aligned}$$

and it follows from equation (4) that

$$\begin{aligned} u_0 &= e\sin(\theta_1 + \theta_2) \\ d\sin\theta_2 &= -e\sin(\theta_1 + \theta_2) \end{aligned}$$

We will refer to the input u_0 associated with an equilibrium point x_0 as the *trim*. It is the DC offset needed to counteract the drift vector field, f, at x_0.

The equilibrium set consists of one or more connected components. In particular, if $d = e$, then we have one connected component, otherwise we have two connected components. These two components consist of equilibrium points where the center of mass of the system is above and below the axis of the first joint, respectively. It is easy to see that if (θ_1, θ_2) is an equilibrium point, then $(-\theta_1, -\theta_2)$ and $(\pi \pm \theta_1, \pi \pm \theta_2)$ are also equlibrium points (see Figure 3).

The kinematic and dynamic parameters for acrobot are given in Table 1. Two sets of values are given. The first corresponds to an acrobot which has an equilibrium set which is a single connected component (i.e., $d = e$). The

Parameter	Units	Balanced Value	Actual Value
l_1	m	1/2	1/2
l_2	m	1	3/4
m_1	kg	8	7
m_2	kg	8	8
g	m/s^2	10	9.8

Table 1: Acrobot parameters. The balanced values correspond to a version of the acrobot which has a connected equilibrium set.

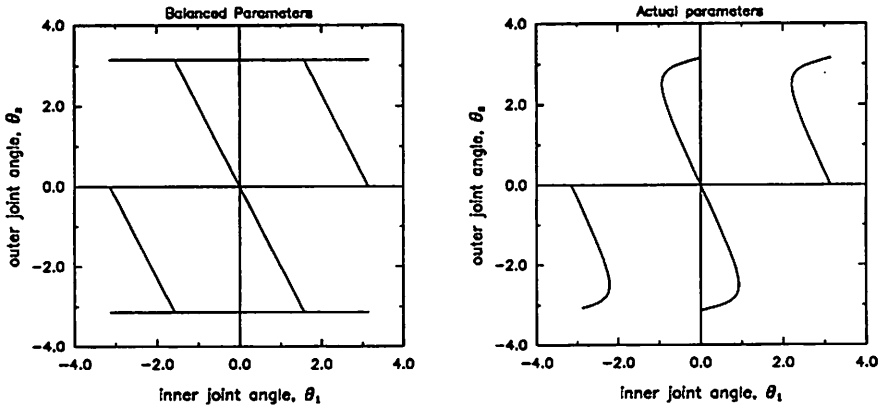

Figure 4: Equilibrium points for acrobot. The left figure is the equilibrium set using the balanced parameter values, the right plot using the actual parameter values.

second set of values is the approximate parameter values for the physical system at U.C. Berkeley. We have rounded units to rational numbers to ease the computational burden. We shall use the former ("balanced") values unless otherwise noted.

The equilibrium points for the two sets of parameters are shown in Figure 4. For the "balanced" parameter values, the equilibrium set consists of all $\theta_1 + \frac{1}{2}\theta_2 = 0$, $\theta_1 + \frac{1}{2}\theta_2 = \pi$, and $\theta_2 = \pm\pi$. This last set of points corresponds to the case where the center of mass of the system is coincident with the axis of the first joint, and hence every value of θ_1 corresponds to an equilibrium configuration. Note also that there is a gap in the range of θ_1 for which the "actual" system may be balanced.

3 Linearization techniques

In this section we explore the application of linearization techniques to the control of the acrobot. We distinguish between two different linearization methods. The first is linearization about a point, in which we approximate the vector fields f and g by their linearizations about an equilibrium point. If the linearization is stabilizable to that equilibrium point, then in a suitably small neighborhood the nonlinear system can also be stabilized (by linear feedback). A more recent technique is feedback linearization (see, for example, Isidori [6]). This method uses a change of coordinates and nonlinear state feedback to transform the nonlinear system description to a linear one (in the new coordinates).

3.1 Linearization about an equilibrium point

If we let $(x_0, u_0) \in \mathbf{R}^4 \times \mathbf{R}$ denote an equilibrium point for the acrobot, the linearization about (x_0, u_0) is given by

$$\dot{z} = Az + bv$$

where

$$\begin{array}{ll} z = x - x_0 & v = u - u_0 \\ A = \frac{\partial}{\partial x}\left(f(x) + g(x)u\right)|_{(x_0,u_0)} & b = g(x_0) \end{array}$$

We refer to this method of linearization as *Jacobian linearization* since it replaces the system vector fields by their Jacobians with respect to x and u evaluated at a point. The linearized system is completely controllable if and only if

$$\det\left[\begin{array}{cccc} b & Ab & \cdots & A^{n-1}b \end{array}\right] \neq 0 \tag{5}$$

It is straightforward to check that the acrobot linearization is completely controllable in a neighborhood of $\theta_1 = \theta_2 = 0$, $\dot{\theta}_1 = \dot{\theta}_2 = 0$ (straight up). At this point

$$A = \left[\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{g}{l_1} & -\frac{gm_2}{l_1 m_1} & 0 & 0 \\ -\frac{g}{l_1} & \frac{g(l_1 m_1 + l_1 m_2 + l_2 m_2)}{l_1 l_2 m_1} & 0 & 0 \end{array}\right] \qquad b = \left[\begin{array}{c} 0 \\ 0 \\ -\frac{l_1 + l_2}{l_1^2 l_2 m_1} \\ \frac{l_1^2 m_1 + (l_1 + l_2)^2 m_2}{l_1^2 l_2^2 m_1 m_2} \end{array}\right]$$

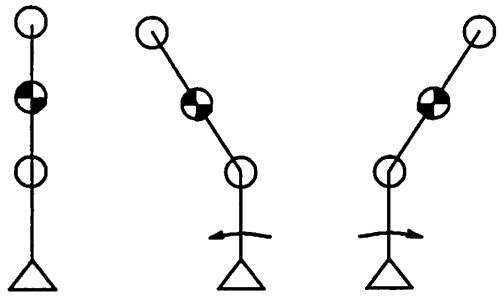

Figure 5: Gravity coupling in the acrobot. By moving the center of mass to one side of the vertical axis, we can cause the entire mechanism to rotate.

By smoothness, it follows that the system is controllable in a neighborhood of the origin. We defer the analysis of points where controllability is lost until later in this section.

Controllability for the acrobot can be given physical interpretation. Consider the case when the mechanism is pointed straight up, with its center of mass directly above the pivot point (see Figure 5). We have direct control of the relative angle of the second link. By moving the second link to the left or right, we can force the center of mass to lie on either side of the pivot point and thus force the whole mechanism to rotate. This use of gravity is crucial in achieving control since equation (5) is not satisfied if $g = 0$ (Ab is zero).

A second effect which occurs is inertial coupling between the first and second links. Since the motor exerts a torque on the second joint relative to the first joint, pushing the second joint in one direction causes the first joint to move in the opposite direction. This phenomenon is seen in the linear model by the presence of a right half plane zero; the transfer function between the hip torque and the angle of the first joint (using the balanced parameter values from Section 2) is:

$$\frac{3\left(s+2\sqrt{\frac{5}{3}}\right)\left(s-2\sqrt{\frac{5}{3}}\right)}{4(s^4-60s^2+400)}$$

Solving for the poles of this transfer function verifies that the acrobot is open loop unstable.

We now return to the question of controllability and investigate equilibrium points at which the linearization is *not* controllable. Figure 6 shows a plot of the determinant of the controllability matrix in equation (5) versus

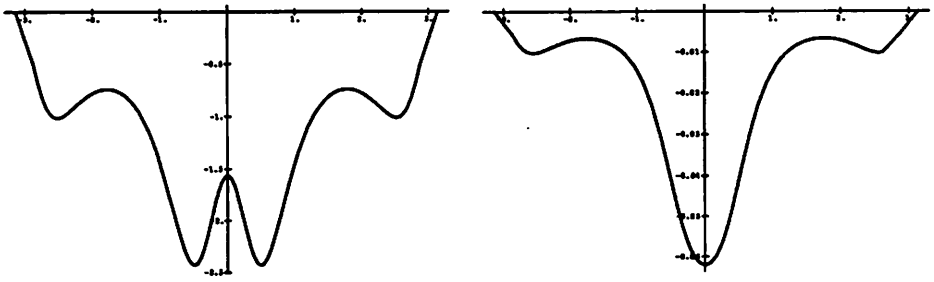

Figure 6: Determinant of controllability matrix versus θ_2. The plot on the left corresponds to the balanced parameters and the plot on the right to the actual parameters.

the hip angle of the acrobot. We see that the system is controllable except at points where $\theta_2 = \pm\pi$. Physically this configuration corresponds to the second link of the acrobot pointing back along the first. In this configuration, the balanced acrobot can swing freely about the axis of the first link and remain in an equilibrium position.

So far our discussion has centered about using a linear controller for stabilization; our real interest is in trajectory tracking. We begin by reviewing trajectory tracking for a linear system

$$\dot{x} = Ax + bu$$

We assume the system is completely controllable and we wish to track a desired state trajectory x_d. Without loss of generality we can assume that (A, b) are in controllable canonical form, i.e. a chain of integrators. In this case the system can be written as

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ &\vdots \\ \dot{x}_{n-1} &= x_n \\ \dot{x}_n &= u \end{aligned}$$

where we have placed all poles at the origin to simplify notation.

If $x^d(\cdot)$ is a desired trajectory which satisfies $\dot{x}^d = Ax^d + bu^d$ for some u^d (i.e., x^d is achievable) then we can follow this trajectory by using

$$u = \dot{x}_n^d$$

when $x^d(0) = x(0)$. This choice of inputs corresponds to injecting the proper input at the end of the chain of integrators which model the system.

To achieve trajectory tracking even if our initial condition does not satisfy $x^d(0) = x(0)$ we introduce the feedback control law

$$u = \dot{x}_n^d + \alpha_1(x_n^d - x_n) + \cdots + \alpha_n(x_1^d - x_1)$$

and the error system satisfies

$$e^{(n)} + \alpha_1 e^{(n-1)} + \cdots + \alpha_n e = 0 \qquad e = x^d - x$$

By choosing the α's so that the resulting transfer function has all of its poles in the left half plane, e will be exponentially stable to 0 and the actual state will converge to the desired state.

In the case of a linearized system, the linearization may not be a good approximation to the system for arbitrary configurations. Since we linearized about a single point, we can only guarantee trajectory tracking in a sufficiently small ball of states about that point. There are several methods for circumventing this problem; one of the most common is gain scheduling. To use gain scheduling, we design tracking controllers for many different equilibrium points and choose our gains based on the equilibrium point(s) to which we are nearest. In fact, this can be done in a more or less continuous fashion using a technique called extended linearization [12]. The basic restriction is that the desired reference trajectory must be slowly varying.

3.2 Feedback linearization

Given a nonlinear system

$$\dot{x} = f(x) + g(x)u \tag{6}$$

it is sometimes possible to find a change of coordinates $\xi = \phi(x)$ and a control law $u = \alpha(x) + \beta(x)v$ such that the resulting dynamics are linear:

$$\dot{\xi} = A\xi + bv$$

In such cases we can control the system by converting the desired trajectory or equilibrium point to our new coordinates, calculating the control v in the that space, and then pulling the control back to the original coordinates. If such a change of coordinates and feedback exists, we say that (6) is *input/state linearizable.*

The conditions under which a general nonlinear system can be converted to a linear one as described above were formulated independently by Jakubcyzk and Respondek and Hunt, Su and Meyer. For the single input case, the conditions are given by the following theorem.

Theorem 1 ([7, 5]) *The system (6) is input/state linearizable in an open set U if and only if*

(i) $\dim \operatorname{span}\{g, ad_f g, \cdots, ad_f^{n-1} g\}(x) = n, \forall x \in U$
(ii) $\operatorname{span}\{g, ad_f g, \cdots, ad_f^{n-2} g\}$ *is an involutive distribution on* U

where $ad_f^j g$ *is the iterated Lie bracket* $[f, \cdots, [f, g] \cdots]$.

The first condition is a controllability test and agrees with the linearization when evaluated at an equilibrium point. The importance of the second condition is more subtle.

If condition (ii) is satisfied, then there exists a smooth $h: \mathbf{R}^n \to \mathbf{R}$ such that

$$\frac{\partial h}{\partial x}\Big[\; g \quad \mathrm{ad}_f g \cdots \mathrm{ad}_f^{n-2} g \;\Big] = 0 \tag{7}$$

This can be seen by applying Frobenius' theorem: since the distribution is involutive, there exists a foliation such that the tangent space to each leaf of the foliation is spanned by the distribution restricted to that leaf. Since the leaves have dimension $n-1$, there exists a scalar valued function h such that the leaves are defined by $h^{-1}(a)$ for $a \in \mathbf{R}$. Equation (7) is essentially saying that the gradient of h is perpendicular to the leaves.

The standard approach in feedback linearization is to use h to define the required change of coordinates. For single input systems we define

$$\begin{aligned} \phi_1(x) &= h(x) \\ \phi_i(x) &= L_f^{i-1} h(x) \end{aligned}$$

where $L_f h = \frac{\partial h}{\partial x} f$ is the Lie derivative of h in the direction f. The condition in equation (7) guarantees that the input will not appear until the n^{th} derivative. Setting $\xi = \phi(x)$, our new equations are

$$\begin{aligned} \dot{\xi}_i &= \xi_{i+1} \qquad\qquad i = 1, \cdots, n-1 \\ \dot{\xi}_n &= a(x) + b(x)u \end{aligned}$$

and by using $u = b^{-1}(-a + v)$ we have a linear system (in Brunowsky canonical form).

Trajectory tracking for such a system is exactly as in the linear case. However, since we have *converted* the model to a linear one instead of approximating it, we do not need to stay close to any particular equilibrium point. Thus in an open set U in which the feedback linearizability equations are satisfied, we can achieve exponential trajectory tracking.

To check the involutivity condition for the acrobot, we must verify that the vector fields

$$[g, \mathrm{ad}_f g] \quad [g, \mathrm{ad}_f^2 g] \quad [\mathrm{ad}_f g, \mathrm{ad}_f^2 g] \tag{8}$$

lie in the distribution

$$\Delta = \mathrm{span}\{g, \mathrm{ad}_f g, \mathrm{ad}_f^2 g\}$$

This can be done by checking that the determinant of a matrix (which is a function of x) is zero. It can be verified (`exact.m`) that the determinant obtained using Δ and the second expression in equation (8) is nonzero. Hence the system is not input/state linearizable.

A less restrictive class of systems is the class of *input/output linearizable* systems. A major difficulty is the possibility of introducing unstable internal dynamics, called *zero dynamics*. Since there is no predefined output function for acrobot, it might be possible to *define* an output such that the system is input/output linearizable and has stable zero dynamics. In this case we could again achieve trajectory tracking by relying on the stable zero dynamics to control unobservable states. Finding such an output function is nontrivial. Both of the obvious output functions (θ_1 and θ_2) have unstable zero dynamics. As we saw with the Jacobian linearization, if we use θ_1 as the output, we obtain a right half plane zero in the linearized system. The effect of this right half plane zero is also present in the nonlinear system. The input/output linearizing feedback cancels this zero with a pole at the same location and results in unstable zero dynamics. Similar problems occur when using θ_2 as the output.

To summarize, we have shown that the acrobot is stabilizable about most equilibrium points (all but a set of measure 0) using static linear state feedback. This simple approach is not suitable for trajectory tracking, although gain scheduling and related approaches might be used to improve performance. The more global method of input/state linearization via state feedback *cannot* be applied to acrobot since the system is provably not input/state linearizable. In the next section we investigate the use of approximate linearization techniques to recover some of the desirable properties of feedback linearization for systems which do not meet the necessary restrictive conditions.

4 Approximate linearization

In the previous section we showed that the acrobot dynamics are not exactly linearizable by state feedback. In this section we apply the technique of approximate linearization to the acrobot. Briefly, we wish to find vector fields $\tilde{f}$ and $\tilde{g}$ which are close to our original vector fields but which satisfy the exact linearizability conditions. We then proceed to design a controller for the approximate system and apply it to the actual system.

The usual method of approximate linearization is slightly complicated in the case of the acrobot for two reasons: we do not have a natural output function and we wish to track trajectories near a manifold of equilibrium points rather than near a single point. This chapter presents a methodology for designing a controller for a system of this type. Briefly, we will proceed in the following manner:

1. Parameterize the controllable equilibrium manifold, $\mathcal{E}$, as $(x_1, 0, \cdots, 0)$.
2. Construct a smooth output, $h(x)$, such that the linearized system at *each* equilibrium point has relative degree n.
3. Using h, construct approximate vector fields $\tilde{f}$ and $\tilde{g}$ such that they approximate f and g along the equilibrium manifold and the approximate system is exactly linearizable.
4. Using $\tilde{f}$ and $\tilde{g}$, design a tracking controller for the approximate system and apply the resulting controller to the original system.

We begin with a brief review of approximation theory using the presentation in Hauser *et. al.* [3] as a guide.

4.1 Review of approximation theory

We consider systems of the form

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned} \tag{9}$$

The system is *input/output linearizable* with relative degree n in a neighborhood U if and only if for all $x \in U$

$$\begin{aligned} &(i) \quad L_g L_f^{i-1} h(x) = 0 \quad i = 1, \cdots, r-1 \\ &(ii) \quad L_g L_f^{n-1} h(x) \neq 0 \end{aligned}$$

where $L_f h = \frac{\partial h}{\partial x} f$ is the Lie derivative of h in the direction f. These conditions are equivalent to the exact linearization conditions in Theorem 1 of the previous section. That is, $\frac{\partial h}{\partial x}$ annihilates the distribution $\{g, \mathrm{ad}_f g, \cdots, \mathrm{ad}_f^{n-2} g\}$. As before, we use the output $\xi = h(x)$ and its first n derivatives to define a new set of coordinates. Using this new set of coordinates, the input/output map is given by the linear transfer function $1/s^n$.

If the input/output conditions are not satisfied, then we can still use this basic construction as a method for generating approximate vector fields which *do* satisfy the conditions, at least in a neighborhood of a controllable equilibrium point. Since the behavior of the nonlinear system about an equilibrium point is determined by its linearization, any approximate system should agree with the linearized system at an equilibrium point (x_0, u_0). That is, the approximate vector field $\tilde{f} + \tilde{g}u$ should agree to first order with the original vector field $f + gu$, when evaluated at the equilibrium point. In particular, this implies that the relative degree of any approximate system should agree with the relative degree of the linearization. This motivates the following definition: the *linearized relative degree* of a nonlinear system in a neighborhood of an equilibrium point x_0 is the relative degree of the linearization about x_0. We use this concept to construct an approximate system which has relative degree equal to the linearized relative degree of the original system.

A key concept is that of *higher order*. A function $\psi(x)$ is said to be higher order at x_0 if the function and its first derivative vanish at x_0. More generally, a function is order k at x_0 if the function and its first k derivatives vanish at x_0, and first order if only the function itself is zero at x_0.

Let x_0 be an equilibrium point of a nonlinear system with u_0 the input required to hold the system at the equilibrium point. Suppose the linearized relative degree of the system about x_0 is n. Then we can define an approximate system in a neighborhood of (x_0, u_0) as follows: set

$$\phi_1(x) = h(x) - \psi_0(x)$$

where ψ_0 is any function that is higher order at x_0. For $i = 2, \cdots, n$, set

$$\phi_i(x) = L_f \phi_{i-1}(x) + u_0 L_g \phi_{i-1}(x) - \psi_{i-1}(x)$$

where $\psi_i(x)$ is higher order at x_0. It can be shown that ϕ is a local diffeomorphism and hence defines a valid change of coordinates. If we write the

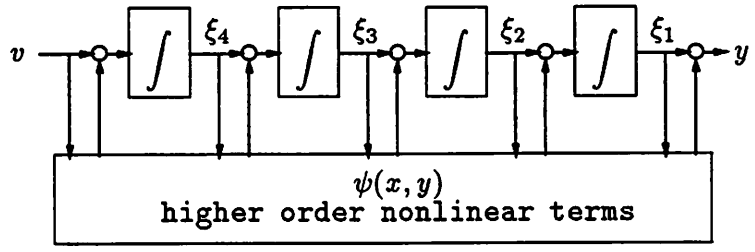

Figure 7: Approximate linearization viewed as a chain of integrators with nonlinear perturbations (from [HKS89]).

system dynamics in this new set of coordinates, we get a chain of integrators with nonlinear perturbations (Figure 4.1).

To see how this procedure produces an approximate system, we pull back the Brunowsky canonical vector fields through the diffeomorphism ϕ to produce the approximate vector fields:

$$\tilde{f} = [D\phi]^{-1} \begin{bmatrix} \phi_2(x) \\ \vdots \\ \phi_n(x) \\ 0 \end{bmatrix} \qquad \tilde{g} = [D\phi]^{-1} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

By construction, the approximate vector fields are input/output linearizable with relative degree n. Furthermore, the vector fields agree with the original vector fields to first order at x_0 since we only throw away higher order terms.

There is a great deal of freedom in choosing the approximation; this freedom is manifested through the choice of the ψ_i's. If the system were input/output linearizable, then we could have chosen ψ_i to be zero at each step and we would have exactly the change of coordinates produced in the exact linearization procedure. Another interesting case is when we choose ψ_i to include *all* second order and higher terms; in this case our approximate system is equivalent to the Jacobian linearization. In general, however, it is not clear which terms to ignore in selecting coordinates. Currently the choice of approximation is a matter of engineering judgement.

Using the approximate system, we can construct an exactly linearizing control law which is capable of trajectory tracking. In our new coordinates,

$\xi = \phi(x)$, the system has the form

$$\begin{aligned} \dot{\xi}_i &= \xi_{i+1} + \psi_i(x) + \theta_i(x)(u - u_0) \\ \dot{\xi}_n &= L_f\phi_n(x) + L_g\phi_n(x)u + \psi_n(x) + \theta_n(x)(u - u_0) \\ y &= \xi_1 + \psi_0(x) \end{aligned}$$

where each θ_i is at least uniformly first order at x_0. With analogy to the exact linearization case, we choose

$$u = \frac{1}{L_g\phi_n(x)}\left(-L_f\phi_n(x) + y_d^{(n)} + \alpha_{n-1}(y_d^{(n-1)} - \xi_n) + \cdots + \alpha_0(y_d - \xi_1)\right) \tag{10}$$

where $s^n + \alpha_{n-1}s^{n-1} + \cdots + \alpha_0$ has all its zeros in the open left half plane. Let

$$\xi_i^d(t) := y_d^{i-1}(t)$$

and define the tracking error as

$$e(t) := \xi^d(t) - \xi(t)$$

This error vector encodes the deviation of the actual system trajectory from the desired trajectory of the approximate system.

For ϵ sufficiently small and desired trajectories which are ϵ-near x_0 and sufficiently slow, the control law (10) results in approximate tracking of the desired trajectory [3]. Thus we can approximately track any trajectory which remains close to the equilibrium point and is slowing varying. A more explicit (and more general) formulation is presented in Section 4.4.

4.2 The equilibrium manifold

In our application, we are not interested in motion near a single equilibrium point, but rather motion near a set of equilibrium points. Given a general single input system, the equilibrium points are those x_0 for which $f(x_0) + g(x_0)u_0 = 0$ for some $u_0 \in \mathbf{R}$. We define $\mathcal{E}$ to be the set of all equilibrium points, x_0, such that the linearized system is controllable about x_0.

Theorem 2 *$\mathcal{E}$ is a manifold of dimension 1.*

Proof. Consider first the set $\mathcal{E}_{x,u}$ of all pairs (x_0, u_0) such that $f(x_0) + g(x_0)u_0 = 0$ and the system is controllable at x_0. Controllability is determined by taking the determinant of a set of smooth functions and hence

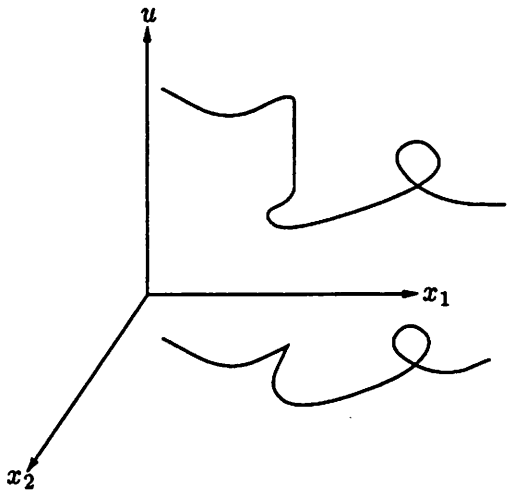

Figure 8: Projection of the equilibrium points onto the state space [10].

there exists and open ball $N \ni (x_0, u_0)$ such that all equilibrium points $(x', u') \in N$ are also controllable. Let U be the union of all such N over $\mathcal{E}_{x,u}$. Then U is open and $\mathcal{E}_{x,u} \subset U$. Define the map $F: U \subset \mathbf{R}^{n+1} \to \mathbf{R}^n$ given by $F: (x, u) \mapsto f(x) + g(x)u$. At any controllable equilibrium point, $F(x_0, u_0) = 0$ and the Jacobian of F,

$$DF(x_0, u_0) = (Df(x_0) + u_0 Dg(x_0), g(x_0)) = (A_0, b_0),$$

is full rank. Hence 0 is a regular value of F and $F^{-1}(0) = \mathcal{E}_{x,u}$ is a submanifold of $\mathbf{R}^n + 1$ of dimension $(n + 1) - n = 1$.

It remains to show that the projection is also a manifold. There are two things that can go wrong: the manifold can be tangent to the projection direction or the manifold can cross over itself. These situations are shown in Figure 8. These singularities can only occur if u_0 cannot be written as a function of x_0. However, at any equilibrium point

$$f(x_0) + g(x_0)u_0 = 0$$

and u_0 is not unique only if $g(x_0) = 0$. This contradicts controllability and hence u_0 is a unique function of x_0 and neither of the situations in Figure 8 can occur. □

We call $\mathcal{E}$ the *controllable equilibrium manifold* and will often refer to it simply as the equilibrium manifold (as opposed to the set of all equilibrium

or operating points). In general $\mathcal{E}$ consists of one or more connected components. For the acrobot there are always two components, consisting of the inverted and non-inverted equilibrium points.

While motion *on* the controllable equilibrium manifold is not possible (since by definition $\dot{x} = 0$ on the manifold), motion *near* the manifold can be achieved. In constructing an approximate system, we wish to do so in a way that keeps the approximation close at equilibrium points. Thus we want to throw away terms which are higher order on the equilibrium manifold (i.e., terms whose value and first derivative vanish on $\mathcal{E}$) while keeping terms that vary along the equilibrium manifold.

In order to construct such an approximation, it is convenient to change coordinates so that the equilibrium manifold has a simple form. A particularly convenient choice of coordinates is one in which points on the equilibrium manifold have the form $(x_1, 0, \cdots, 0)$. We can always find a parameterization of the equilibrium manifold which has this form in a neighborhood of a controllable equilibrium point, since $\mathcal{E}$ is a one dimensional manifold.

For the acrobot, we have chosen to parameterize the equilibrium manifold using the hip angle. For the second configuration variable we use the angle of the center of mass of the system—this must be zero at all inverted equilibrium points since the center of mass must lie directly above the axis of the first link. We complete the state with the velocities of the two configuration variables. These calculations are contained in (`equilibrium.m`). The resulting change of coordinates (see Appendix A) is:

$$\begin{aligned} x_1 &= \theta_2 \\ x_2 &= \theta_1 + \frac{e \sin \theta_2}{\sqrt{d^2 + e^2 + 2ed \cos \theta_2}} \\ x_3 &= \dot{x}_1 \\ x_4 &= \dot{x}_2 \end{aligned}$$

Other parameterizations are possible. For example, one might choose the x and y components of the system center of mass as the configuration variables. Unfortunately, the parameterization is singular about the straight up position, just as it is for a two-link robot manipulator. Another advantage of the parameterization we chose is that it simplifies some of the calculations. In particular, for the balanced system parameters mentioned in Section 2, the angle of the center of mass is simply $\theta_1 + \frac{1}{2}\theta_2$ whereas x_{cm} and y_{cm} involve trigonometric functions. This is the original motivation for defining the "balanced" set of parameters.

4.3 Constructing an (artificial) output function

In the approximation theory presented above, an output function was used to construct the approximate system. In some applications, the system possesses a natural output function that can be used for this purpose. However, in the case of the acrobot, no suitable output function is given so we must *construct* one. In this section we present a technique for doing so. As usual, we begin by considering the linear case.

Suppose we are given a controllable linear system

$$\dot{x} = Ax + bu$$

and we are asked to find an output

$$y = cx$$

which is suitable for stable trajectory tracking. By this we mean that it is easy to design a controller to make $y(t)$ track a desired trajectory $y_d(t)$ while maintaining internal stability of the system. If the system is in Brunowsky form (i.e., a chain of integrators), then a natural output function is the output from the last integrator. This insures that the system has no zeros so that $y(t), \dot{y}(t), \ldots, y^{(n-1)}(t)$ can be used as the n coordinates of the system state. In particular, if the output $y(t)$ converges to a constant value, then the system will converge to an equilibrium point.

To construct this output when the system (A, b) is not in Brunowsky canonical form, we note that the relative degree of the system is given by the largest r such that

$$\begin{aligned} cA^{i-1}b &= 0 \qquad i = 1, \cdots, r-1 \\ cA^{r-1}b &\neq 0 \end{aligned}$$

Since we want the relative degree to be n (no zeros), we require that

$$c\begin{bmatrix} b & Ab & \cdots & A^{n-2}b \end{bmatrix} = 0. \tag{11}$$

Thus, any $c \neq 0$ in the (1-dimensional) null space indicated by equation (11) defines an output such that the system (c, A, b) has relative degree n.

We now return to the nonlinear system

$$\dot{x} = f(x) + g(x)u$$

with the goal of constructing an output

$$y = h(x)$$

to use in constructing an approximate system for control design. If the system with output is input/output linearizable with relative degree n around x_0 then the system is linearly controllable and satisfies the nonlinear analog to (11) given by

$$\frac{\partial h}{\partial x}\Big[\; g \quad \mathrm{ad}_f g \quad \cdots \quad \mathrm{ad}_f^{n-2} g \;\Big] = 0$$

for all x in a neighborhood of x_0. In other words, the system is input/state linearizable—it satisfies the conditions of Theorem 1. Since many systems such as the acrobot are not input/state linearizable, we look to approximation. Our problem is one of finding a function h and approximate vector fields $\tilde{f}$ and $\tilde{g}$ such that

$$\frac{\partial h}{\partial x}\Big[\; \tilde{g} \quad \mathrm{ad}_{\tilde{f}} \tilde{g} \quad \cdots \quad \mathrm{ad}_{\tilde{f}}^{n-2} \tilde{g} \;\Big] = 0 \tag{12}$$

for all x in a neighborhood of x_0 or, more generally, in a neighborhood of the equilibrium manifold.

Since it is extremely difficult to directly modify the vector fields f, g so that the system is exactly input/state linearizable, we will first construct the *output* function h and then use the approximate linearization methodology to construct $\tilde{f}$ and $\tilde{g}$. The basic idea is to find a function h that satisfies equation (12) at each point on the equilibrium manifold. Provided that the original and approximate systems agree to first order on the equilibrium manifold, the *ad*-chains of the two systems will span the same subspace at each point on the equilibrium manifold, that is,

$$\mathrm{span}\{g, \mathrm{ad}_f g, \cdots, \mathrm{ad}_f^{n-2} g\} = \mathrm{span}\{\tilde{g}, \mathrm{ad}_{\tilde{f}} \tilde{g}, \cdots, \mathrm{ad}_{\tilde{f}}^{n-2} \tilde{g}\}$$

for $x \in \mathcal{E}$. In fact, these calculations can be done directly with the *linearization* of the *original* system on the equilibrium manifold. This point is somewhat subtle, so we describe it in detail.

We will assume that coordinates have been chosen such that the equilibrium manifold $\mathcal{E}$ has been straightened out so that each $x \in \mathcal{E}$ has the form $(x_1, 0, \ldots, 0)$. Let $x_e(x_1)$ and $u_e(x_1)$ denote the state and control for each equilibrium point $(x_1, 0, \ldots, 0)$ on $\mathcal{E}$, that is,

$$x_e(x_1) = (x_1, 0, \ldots, 0)$$

and $u_e(\cdot)$ is such that

$$f(x_e(x_1)) + g(x_e(x_1))u_e(x_1) = 0$$

for each x_1 such that $x_e(x_1) \in \mathcal{E}$.

Suppose, at first, that we *trim* the drift vector field

$$\bar{f}(x) := f(x) + g(x)u_c(x)$$

where $u_c(\cdot)$ is any control satisfying $u_c(x_e(x_1)) = u_e(x_1)$. The linearization of the trimmed system along the equilibrium manifold is then given by

$$\dot{z} = \bar{A}(x_1)z + b(x_1)v$$

where

$$\begin{array}{rcl} \bar{A}(x_1) & := & \frac{\partial f}{\partial x}(x_e(x_1)) + u_c(x_e(x_1))\frac{\partial g}{\partial x}(x_e(x_1)) + g(x_e(x_1))\frac{\partial u_c}{\partial x}(x_e(x_1)) \\ & = & \frac{\partial f}{\partial x}(x_e(x_1)) + u_e(x_1)\frac{\partial g}{\partial x}(x_e(x_1)) + g(x_e(x_1))\frac{\partial u_c}{\partial x}(x_e(x_1)) \\ b(x_1) & := & g(x_e(x_1)) \end{array}$$

In this case it is easy to verify that

$$\mathrm{ad}_f^j g(x_e(x_1)) = (-\bar{A}(x_1))^j b(x_1)$$

Thus, letting $c(\cdot)$ be the derivative of the yet to be constructed output function h along the equilibrium manifold,

$$c(x_1) := \frac{\partial h}{\partial x}(x_e(x_1)),$$

equation (12) (for the trimmed system) evaluated along $\mathcal{E}$ takes the form

$$c(x_1)\Big[\; b(x_1) \quad \bar{A}(x_1)b(x_1) \quad \cdots \quad \bar{A}(x_1)^{n-2}b(x_1) \;\Big] = 0 \qquad (13)$$

The equation has a smooth solution $c(\cdot)$ on $\mathcal{E}$ since the system is, by definition, linearly controllable at each of these points. Unfortunately, this linearization depends on the choice of the trim function $u_c(\cdot)$. Certainly, one does not expect that the choice of the trim function can materially affect the directions in which the system can be controlled. Additionally, since we plan to do symbolic calculations to construct the output function, we seek the simplest expressions for these objects.

Note that the actual trim $u_e(x_1)$ needed at an equilibrium point is uniquely defined. If, at a given equilibrium point $x_e(x_1)$ we freeze the trimming control $u_c(x) \equiv u_e(x_1)$ then the linearization will be given by

$$\dot{z} = A(x_1)z + b(x_1)v$$

where

$$A(x_1) \;:=\; \tfrac{\partial f}{\partial x}(x_e(x_1)) + u_e(x_1)\tfrac{\partial g}{\partial x}(x_e(x_1))$$

Note that $A(x_1) \neq \bar{A}(x_1)$ due to the presence of the $\frac{\partial u_c}{\partial x}$ term. In fact,

$$\bar{A}(x_1) = A(x_1) + b(x_1)\frac{\partial u_c}{\partial x}(x_e(x_1))$$

The following lemma shows that we can use the well-defined expression $A(\cdot)$ for our calculations in place of the somewhat arbitrary expression $\bar{A}(\cdot)$.

Lemma 1 *Given $A(\cdot)$, $b(\cdot)$, and $\bar{A}(\cdot)$ as defined above,*

$$\text{span}\{b(x_1), \cdots, A(x_1)^{j-2}b(x_1)\} = \text{span}\{b(x_1), \cdots, \bar{A}(x_1)^{j-2}b(x_1)\}$$

for $j = 2, 3, \ldots$.

Proof. The lemma is trivially true if $j = 2$. Suppose the lemma holds for $j \leq k$.

$$\begin{aligned}\bar{A}(x_1)^{k+1}b &= \left(A(x_1) + g(x_e(x_1))\tfrac{\partial u_c}{\partial x}(x_e(x_1))\right)\bar{A}(x_1)^k b(x_1) \\ &= A(x_1)\bar{A}(x_1)^k b(x_1) + b(x_1)[\tfrac{\partial u_c}{\partial x}(x_e(x_1))\bar{A}(x_1)^k b(x_1)]\end{aligned}$$

The first term is contained in $\text{span}\{b(x_1), \cdots, A(x_1)^{k+1}b(x_1)\}$ since

$$\bar{A}(x_1)^k b(x_1) \in \text{span}\{b(x_1), \cdots, A(x_1)^k b(x_1)\}$$

The second term is a multiple of $b(x_1)$ and hence it is also in

$$\text{span}\{b(x_1), \cdots, A(x_1)^{k+1}b(x_1)\}.$$

□

Thus we see that the derivative $c(\cdot)$ of our output function $h(\cdot)$ solves the equation

$$c(x_1)\Big[\; b(x_1) \quad A(x_1)b_0 \quad \cdots \quad A(x_1)^{n-2}b(x_1) \;\Big] = 0 \qquad (14)$$

It is clear that $c(x_1) = (c_1(x_1), \cdots, c_n(x_1))$ (viewed as a differential one-form) is integrable. Indeed, we integrate

$$dh(x) = c_1(x_1)dx_1 + \cdots + c_n(x_1)dx_n$$

to get

$$h(x) = \int c_1(x_1)dx_1 + c_2(x_1)x_2 + \cdots + c_n(x_1)x_n$$

Further, since x_1 parameterizes the equilibrium manifold, we have the following useful fact:

Lemma 2 *Suppose that $c(x_1) \neq 0$ solves (14) with $x_e(x_1) \in \mathcal{E}$. Then $c_1(x_1) \neq 0$.*

Proof. By Lemma 1, we may assume that $f(x) = 0$ for $x \in \mathcal{E}$. Since the system is linearly controllable on $\mathcal{E}$, the vectors

$$\{b(x_1), A(x_1)b(x_1), \cdots, A(x_1)^{n-2}b(x_1)\}$$

are linearly independent and $c(x_1)$ lies in the left null space of these vectors. It suffices to show that $e^1 = (1, 0, \cdots, 0)^T$ is linearly independent of these vectors since this implies $c_1(x_1) = c(x_1) \cdot e^1 \neq 0$. But we see that

$$A(x_1) \cdot e^1 = \frac{\partial f}{\partial x_1}(x)\Big|_{x_e(x_1)}$$

and this last expression is zero since since $f(x) \equiv 0$ along the equilibrium manifold, parameterized by x_1. Hence e^1 is in the null space of A_0 and the vectors $b_0, A_0b_0, \cdots, A_0^{n-2}b_0$ are not in the null space of A_0 since

$$\{A_0b_0, \cdots, A_0^{n-1}b_0\}$$

are also linearly independent by the controllability assumption. Therefore e^1 is linearly independent of $\{b_0, A_0b_0, \cdots, A_0^{n-2}b_0\}$ and $c_1(x_0) = c_0 \cdot e^1 \neq 0$. □

Given this fact, we can write

$$\begin{aligned} dh(x) &= c_1(x_1)dx_1 + c_2(x_1)dx_2 + \cdots + c_n(x_1)dx_n \\ &= dx_1 + \tfrac{c_2(x_1)}{c_1(x_1)}dx_2 + \cdots + \tfrac{c_n(x_1)}{c_1(x_1)}dx_n \\ &= dx_1 + \tilde{c}_2(x_1)dx_2 + \cdots + \tilde{c}_n(x_1)dx_n \\ h(x) &= x_1 + \tilde{c}_2(x_1)x_2 + \cdots + \tilde{c}_n(x_1)x_n \end{aligned}$$

Any h which matches this expression to first order is also a valid output function, with linear relative degree n. For the acrobot, the output function which results from the above calculation is (`output.m`)

$$h(x) = x_1 + (6 + 4\cos x_1)x_2$$

4.4 Approximate tracking near an equilibrium manifold

We can now extend the approximation procedure presented in Section 4.1 to construct a controller which tracks slowly varying trajectories near an equilibrium manifold. To do so, we extend the concept of a higher order function. We say a function is *uniformly higher order* on a manifold (parameterized by x_1) if it is higher order in $(x_2, \cdots, x_n)$. Thus in the approximation procedure, we will ignore terms which are small near the equilibrium manifold, while keeping terms that vary along the manifold. This section details that procedure and concludes with a proof of approximate tracking for control laws constructed in this manner.

It will be convenient at this point to assume that $f(x_0) = 0$ for $x_0 \in \mathcal{E}$. Although we took pains to avoid making this assumption in the previous section, the benefit of allowing $f(x_0) \neq 0$ is outweighed here by a tremendous increase in notation. We therefore assume that any nonlinear trim is included in the drift vector field. This can be accomplished in many ways, the simplist of which is to define

$$\begin{aligned} \bar{f}(x) &= f(x) - g(x)u_e(x_1) \\ v &= u - u_e(x_1) \end{aligned}$$

and write our system as

$$\begin{aligned} \dot{x} &= \bar{f}(x) + g(x)v \\ y &= h(x) \end{aligned}$$

Suppose the linearized relative degree of the system $(\bar{f}, g)$ with respect to an output function h is n on an equilibrium manifold $\mathcal{E} = \{(x_1, 0, \cdots, 0)\}$. Assuming $\bar{f}(x_0) = 0$ for $x_0 \in \mathcal{E}$, we define a new set of coordinates $\xi = \phi(x) \in \mathbf{R}^n$:

$$\begin{aligned} \phi_1(x) &= h(x) - \psi_0(x) \\ \phi_2(x) &= L_{\bar{f}}\phi_1(x) - \psi_1(x) \\ &\vdots \\ \phi_n(x) &= L_{\bar{f}}\phi_{n-1}(x) - \psi_{n-1}(x) \end{aligned}$$

where each $\psi_i(x)$ is uniformly higher order on $\mathcal{E}$. The system dynamics in ξ

coordinates are

$$\begin{aligned} \dot{\xi}_1 &= \xi_2 + \psi_1(x) + \theta_1(x)v \\ &\vdots \\ \dot{\xi}_{n-1} &= \xi_n + \psi_{n-1}(x) + \theta_{n-1}(x)v \\ \dot{\xi}_n &= L_f\phi_n(x) + L_g\phi_n(x)v + \psi_n(x) + \theta_n(x)v \\ \\ y &= \xi_1 + \psi_0(x) \end{aligned} \tag{15}$$

where each $\theta_i(x)$ is at least uniformly first order on $\mathcal{E}$. As in the previous approximation procedure, the choice of ψ allows considerable freedom in constructing the approximation. Since the linearization is controllable on $\mathcal{E}$ and $\frac{\partial h}{\partial x}(x)$ satisfies (14), it follows that $L_g\phi_n(x_0) \neq 0$ for $x_0 \in \mathcal{E}$.

Because the functions ψ_i are uniformly higher order on $\mathcal{E}$ and the functions θ_i are at least uniformly first order on $\mathcal{E}$, the approximate system

$$\begin{aligned} \dot{\xi}_1 &= \xi_2 \\ &\vdots \\ \dot{\xi}_{n-1} &= \xi_n \\ \dot{\xi}_n &= L_f\phi_n(x) + L_g\phi_n(x)v \\ \\ y &= \xi_1 \end{aligned} \tag{16}$$

is a uniform system approximation of (f, g) on $\mathcal{E}$ [4]. To provide approximate tracking control for the true system (15), we will use the exact asymptotic tracking control law for the approximate system (16), namely,

$$v = \frac{1}{L_g\phi_n(x)}\left[-L_f\phi_n(x) + y_d^{(n)}(t) + \sum_{i=0}^{n-1}\alpha_i(y_d^{(i)} - \phi_{i+1}(x))\right] \tag{17}$$

where

$$s^n + \alpha_{n-1}s^{n-1} + \cdots + \alpha_1 s + \alpha_0 \tag{18}$$

is a Hurwitz polynomial. As before, we define $\xi^d(t)$ to be the state trajectory for the approximate system induced by the desired output, $y_d(\cdot)$,

$$\xi_i^d(t) := y_d^{(i-1)}(t)$$

We then expect that the tracking error

$$e(t) := \xi^d(t) - \xi(t)$$

will remain bounded for reasonable trajectories. In fact, we will see that the size of the tracking error will be influenced by how far the desired trajectory strays from the equilibrium manifold.

Since the approximate system (16) is a uniform system approximation of the true system (15) around $\mathcal{E}$, we would expect that the approximation would be valid on, for instance, a cylindrical neighborhood of $\mathcal{E}$ given by

$$C_\epsilon(\mathcal{E}) := \{\xi : \pi_1\xi \in \mathcal{E}, \|\xi - \pi_1\xi\| < \epsilon\}$$

where $\pi_1\xi := (\xi_1, 0, \ldots, 0)$ and ϵ is sufficiently small. We make use of the following fact: it is always possible to choose $\mathcal{E}' \subset \mathcal{E}$ so that a given function $\lambda(\xi)$ that is uniformly order ρ on $\mathcal{E}$ will satisfy

$$|\lambda(\xi)| < K\|\xi - \pi_1\xi\|^\rho$$

for all $\xi \in C_\epsilon(\mathcal{E}')$, $0 < \epsilon < 1$. For example, let $\lambda(\xi) = \xi_1\xi_2^2$. Choosing $\mathcal{E}' = \{\xi \in \Re^2 : |\xi_1| < K, \xi_2 = 0\}$ will guarantee the $\lambda(\xi) < K\xi_2^2$ on $C_\epsilon(\mathcal{E}')$, $0 < \epsilon < 1$.

The following theorem shows that such a control law can indeed provide the desired result and provide stable approximate tracking in the neighborhood of the equilibrium manifold.

Theorem 3 *Suppose (f, g) is linearly controllable at x_0 and let $\mathcal{E}$ be the manifold of linearly controllable equilibrium points. Further assume that $f(x_e) = 0$ for $x_e \in \mathcal{E}$. Then, there exists a manifold $\mathcal{E}' \subset \mathcal{E}$, a change of coordinates $\xi = \phi(x)$, and an $\epsilon > 0$ such that the approximate tracking control law (17) results in stable approximate tracking provided $\xi^d(t) \in C_\epsilon(\mathcal{E}')$ and $|y_d^{(n)}(t)| \leq \epsilon$ for $t \geq 0$, and $\|e(0)\| \leq \epsilon$. Furthermore, the tracking error will be of order ϵ^2.*

Proof. Construct a system approximation as detailed above. For convenience, define

$$\begin{aligned} \psi(\xi) &= (\psi_1(x), \cdots, \psi_n(x))|_{x=\phi^{-1}(x)} \\ \theta(\xi) &= (\theta_1(x), \cdots, \theta_n(x))|_{x=\phi^{-1}(x)} \end{aligned}$$

The closed loop system given by (15) and (17) can be written as

$$\dot{e} = Ae + \psi(\xi) + \theta(\xi)v$$

where A is a Hurwitz matrix with characteristic polynomial (18).

As discussed above, we may take $\mathcal{E}'$ to be such that

$$\begin{aligned} \|\psi(\xi)\| &\leq k_1\|\xi - \pi_1\xi\|^2 \\ \|\theta(\xi)\| &\leq k_1\|\xi - \pi_1\xi\| \\ \|L_{\bar{f}}\phi_n(\xi)\| &\leq k_1\|\xi - \pi_1\xi\| \end{aligned}$$

for $\xi \in C_\delta(\mathcal{E}')$, $\delta < 1$, and some $k_1 < \infty$. Since $L_g\phi_n(x)$ is nonzero on $\mathcal{E}$, we can also require that $\mathcal{E}'$ and δ be such that

$$\left|\frac{1}{L_g\phi_n(\xi)}\right| < k_2$$

for $\xi \in C_\delta(\mathcal{E}')$ and some $k_2 < \infty$. Using these bounds plus the fact that

$$\|\xi - \pi_1\xi\| \leq \|e\| + \epsilon$$

(by choice of $y_d(\cdot)$), it follows that there exists $k_3 < \infty$ such that

$$\|\psi(\xi) + \theta(\xi)v\| \leq k_3(\|e\|^2 + \epsilon\|e\| + \epsilon^2)$$

where $\xi \in C_\delta(\mathcal{E}')$.

Choose the Lyapunov function

$$V = e^T P e$$

where $P > 0$ solves $A^TP + PA = -I$. Differentiating V along the trajectories of the closed loop system, for $\xi \in C_\delta(\mathcal{E}')$ and some $k_4 < \infty$,

$$\begin{aligned} \dot{V} &= -\|e\|^2 + 2e^TP(\psi(\xi) + \theta(\xi)(u - u_0(\xi)) \\ &\leq -\|e\|^2 + k_4\|e\|(\|e\|^2 + \epsilon\|e\| + \epsilon^2) \\ &\leq -\tfrac{1}{4}\|e\|^2 - (\tfrac{1}{2} - k_4(\|e\| + \epsilon))\|e\|^2 - (\tfrac{1}{2}\|e\| - k_4\epsilon^2)^2 + k_4^2\epsilon^4 \end{aligned}$$

If $\|e\| \leq \frac{1}{2k_4} - \epsilon$, we have

$$\dot{V} < -\frac{1}{4}\|e\|^2 + k_4^2\epsilon^4.$$

and hence $\dot{V}$ is strictly negative whenever $2k_4\epsilon^2 \leq \|e\| \leq \frac{1}{2k_4} - \epsilon$. By making ϵ sufficiently small, we can guarantee that $e(t)$ will converge to a ball of order ϵ^2 for all $\|e(0)\|$ sufficiently small. Note that the above analysis is valid since

$$\|\xi - \pi_1\xi\| \leq \epsilon + \sup\|e(t)\| < \delta < 1$$

is satisfied when ϵ and $\|e(0)\|$ are sufficiently small, and hence $\xi(t) \in C_\delta(\mathcal{E}')$ under these conditions. □

Corollary 3.1 *If there is a time $t_1 \geq 0$ such that the desired output trajectory becomes constant, i.e., $y_d(t) \equiv y_1$, $t \geq t_1$, then the trajectory tracking error $e(t)$ will converge to zero and the system will converge to the constant operating point $\xi = (y_1, 0, \ldots, 0)$.*

As we mentioned above, it is possible to extend this analysis such that $f(x_0) = 0$, $x_0 \in \mathcal{E}$ is not required. Although removing this assumption can unnecessarily complicate the analysis, there is one special case which is illuminating. If we choose a change of coordinates such that u never appears in the derivatives $L_{f+gu}\phi_i$, we do not need to assume that $f(x_0) = 0$. In this special case we can choose

$$\phi_i(x) = L_f\phi_{i-1}(x) - \psi_{i-1}(x)$$

and no θ_{i-1} term appears in the corresponding $\dot{\xi}_i$ since the input does not appear (by choice of ψ). It turns out that for the approximations constructed for the acrobot, the input never appears and hence we can make use of this simplification and avoid the additional computational burden assocaited with calculations involving $\bar{f} = f + gu_0$. It is important to note that this simplification is not generic and may fail to hold for specific systems.

The next chapter gives details on the results of applying this controller formulation to the acrobot.

5 Controller comparisons and discussion

In this section we present comparisons of a linear and nonlinear controllers for acrobot. We present three controllers, representing various system approximations: linearization about a point, linearization about the equilibrium manifold, and uniformly higher order approximation. In order to properly adjust for gains, we have in all cases converted the systems into (approximate) Brunowsky canonical form and then applied the appropriate design criteria. The output function for each controller is the one derived in Section 4.3, which gives linearized relative degree $n = 4$ along the equilibrium manifold. Also, except as noted, we have used the special set of parameters for acrobot which makes the equilibrium coordinates trivial. For simplicity, we refer to the controllers as linear, gain-scheduled, and nonlinear.

The linear controller was constructed by linearizing the acrobot about the completely inverted position, $\theta_1 = \theta_2 = 0$ (`linear.m`). This configuration is roughly in the center of the operating region which we considered. The controller is implemented as a linear tracking controller (see Section 3.1) using "balancing" coordinates (i.e., the equilibrium manifold is parameterized by x_1).

The gain-scheduled controller is similar to the linear controller, except that all calculations are carried out as a function of x_1, the projection of the state onto the equilibrium manifold (`schedule.m`). The controller is constructed using a change of coordinates which ignores all second order and higher nonlinearities in the variables x_2, x_3, x_4. In that set of coordinates we choose the gains to set the pole locations appropriately. This controller is similar to the controllers described by [10, 12].

The nonlinear controller is constructed using a change of coordinates which throws away higher order terms (`approximate.m`) in the system velocities, x_3, x_4. Thus terms of the form $x_3 \sin x_2$ are not thrown away in this approximation. Furthermore, *all* nonlinearities are kept in the calculations of $L_f \phi_n$ and $L_g \phi_n$.

The gains for each controller were chosen using the same design criteria. We placed all poles of the (approximating) closed loop system at -3.5. This choice represented a compromise between performance and stability. Because the acrobot is operating in an inverted position, large overshoots can move the state out of the region of stability. Other pole locations have been tested, but are not presented here.

All simulations were generated using a Mathematica simulation package that converted system descriptions into C source and generated an exe-

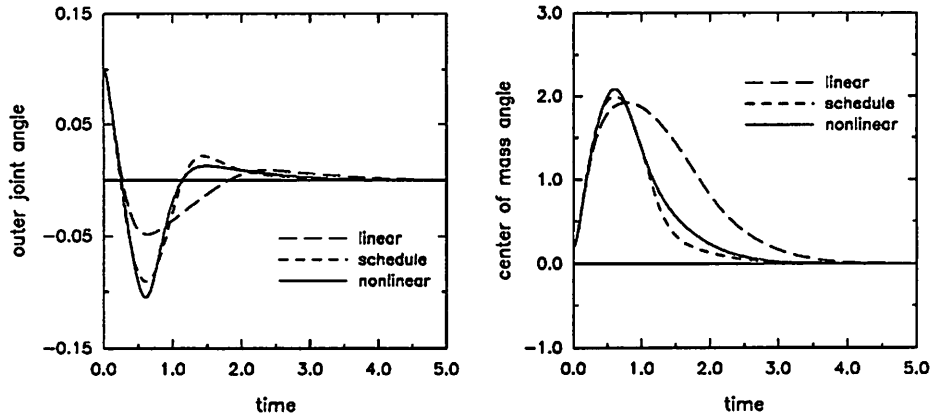

Figure 9: Stability comparison. The left plot shows the ange of the second joint, $x_1 = \theta_2$. The right plot showls the angle of the center of mass of the system, $x_2 = \theta_1 + \frac{1}{2}\theta_2$.

cutable simulation program. A variable step size Runge-Kutta integrator was used to integrate trajectories.

5.1 Stabilizing controllers

For regulation to an equilibrium point, the system performance is similar for all three controllers (`stability.m`). The region of attraction is not noticeably different though the linear system converges somewhat more slowly. This is due to the fact that the linear controller sees a reduced effective gain at system configurations away from the nominal operating point. In contrast, the nonlinear controllers provide *instantaneous gain scheduling* at each position near the equilibrium manifold. This phenomenon is clearly shown in figure 9 where the initial position was given by $\theta_1 = 0$, $\theta_2 = .2$ and regulation to $\theta_1 = \theta_2 = 0$ was desired.

A slice of the region of stability is shown in Figure 10. This slice shows the set of initial conditions with $\dot{\theta}_1 = \dot{\theta}_2 = 0$ which converged to the origin. The region of stability is roughly uniform size about the equilibrium manifold.

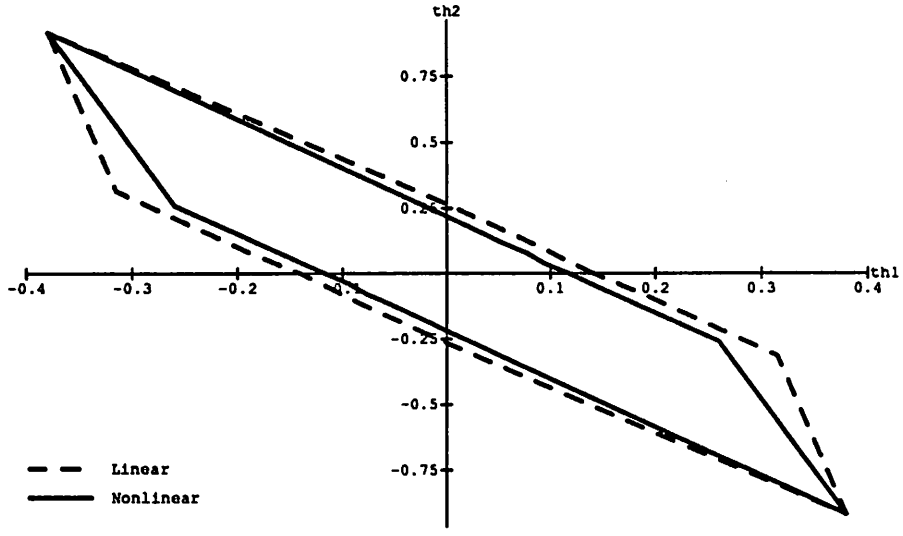

Figure 10: Region of attraction ($\dot{\theta} = 0$ slice)

5.2 Tracking controllers

A more striking difference in controller performance is apparent when we attempt to track a trajectory (`tracking.m`). As evident from Figure 11, the nonlinear controllers had significantly better output tracking capability. A large part of the linear controller error results because a strictly linear controller cannot calculate the input necessary to hold the nonlinear system at more than one operating point (this requires a nonlinear function or table lookup). The nonlinear controllers, however, directly provide the instantaneous *nonlinear trim* needed at each different system configuration along the equilibrium manifold.

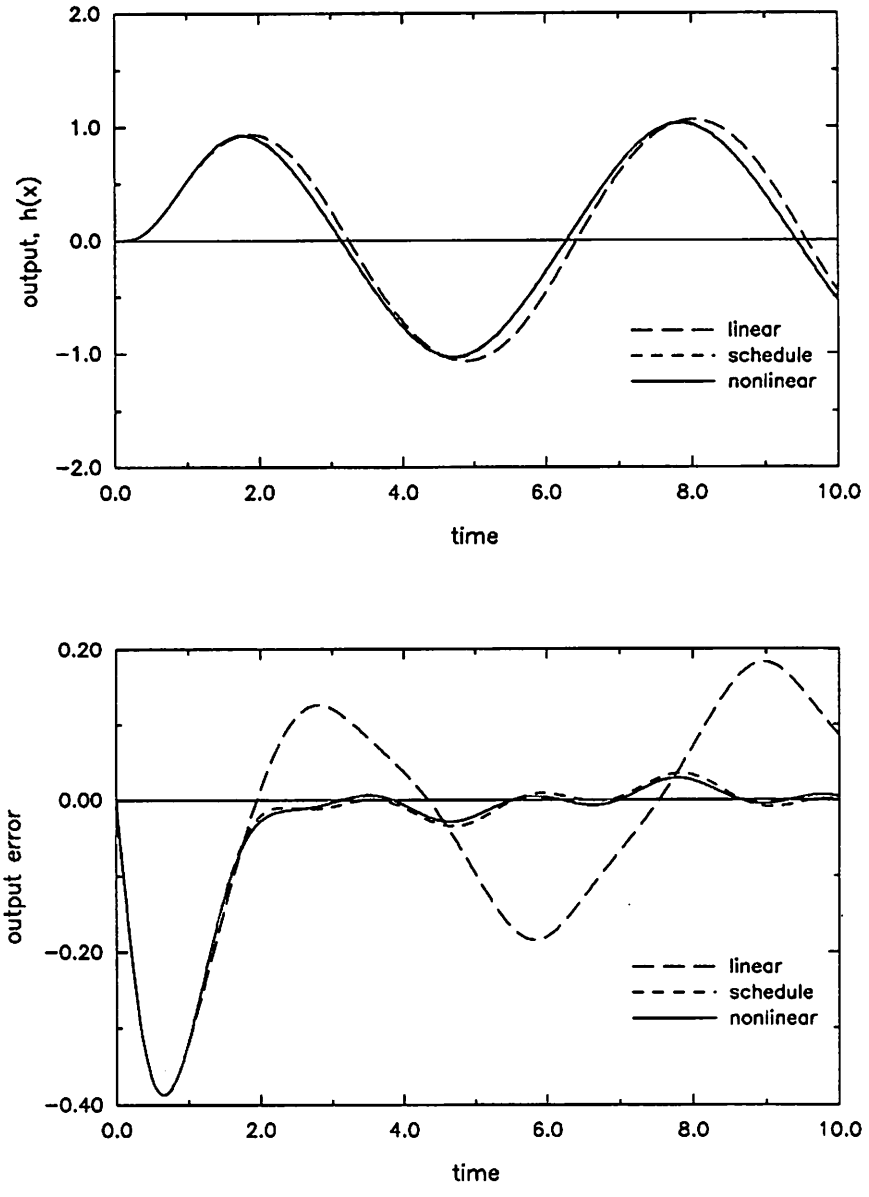

Figure 11: Tracking comparison. The upper plot shows the value of the output function, $\xi_1 = h(x)$, as it tracks a sinusoidal reference trajectory of magnitude 1 and frequency $\omega = 1$ rad/sec. The lower plot shols the error between the desired and actual output trajectories.

5.3 Tracking with the UCB acrobot parameters

Figure 12 shows a comparison of the three controllers using the parameter values associated with the UC Berkeley acrobot (see Table 1). For this set of parameters, the equilibrium set has two distinct components.

For the simulation in Figure 12 it is not clear that the nonlinear controller is improving the tracking error. However, if we slow down the desired trajectory, the improvement is more apparent, as shown in Figure 13. This improvement is not unexpected, since one of the conditions of the Theorem 3 was that the trajectory be slowly varying.

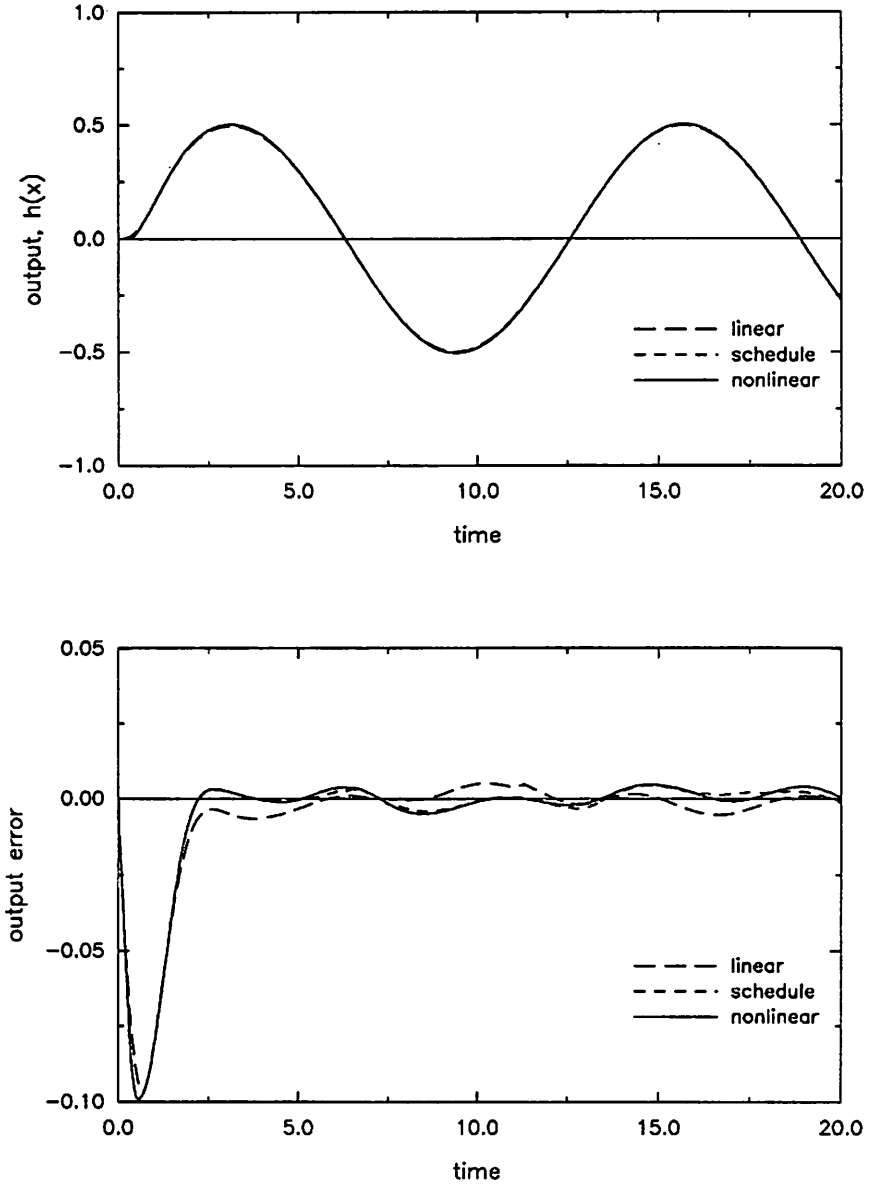

Figure 12: Tracking comparison for UCB acrobot parameters using a fast trajectory. The upper plot shols the output trajectory and the lower plot shows the output error. The frequency of the reference trajectroy is ω = 0.5 rad/sec.

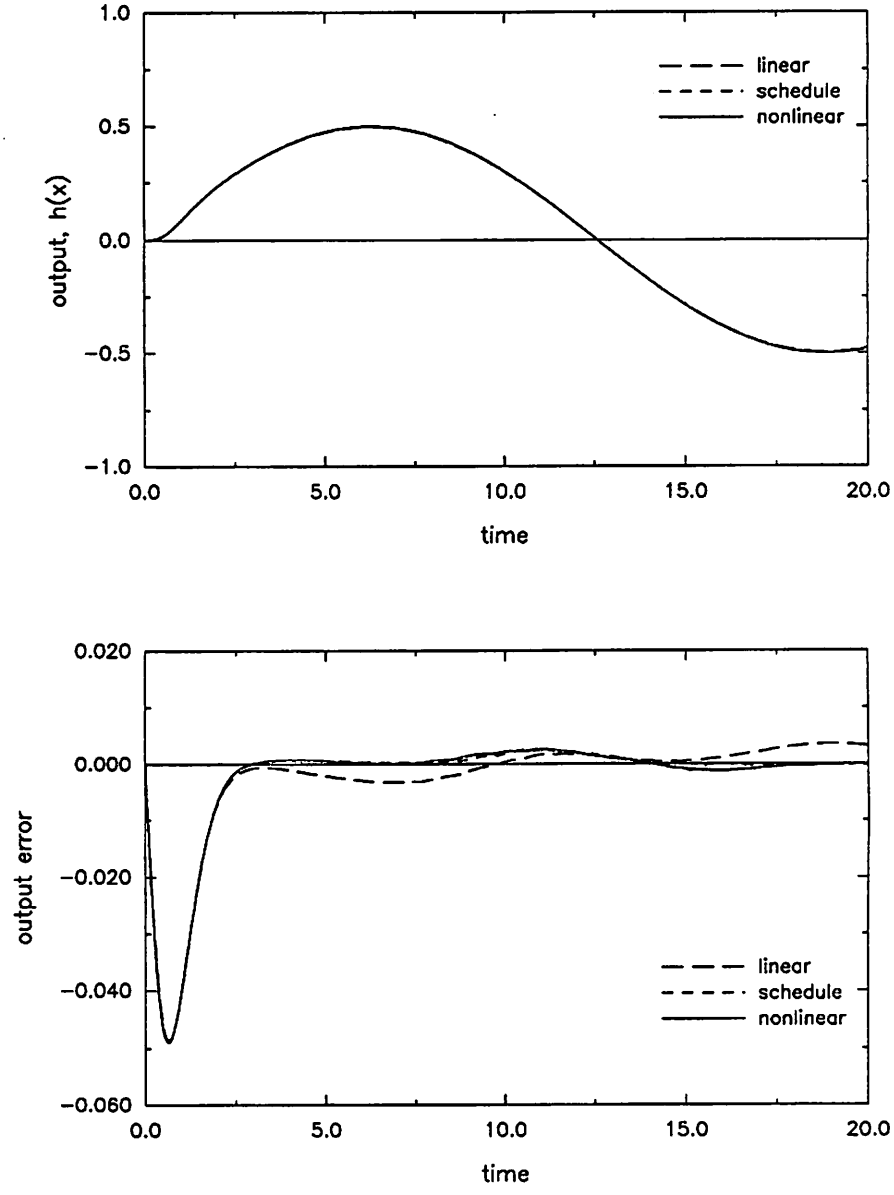

Figure 13: Tracking comparison for UCB acrobot parameters using a slow trajectory. The frequency of the reference trajectory is $\omega = 0.25$ rad/sec.

5.4 Discussion

The acrobot is an example of a system which violates many of the usual assumptions which are required for defining nonlinear control laws. In particular, there is no natural output function and the system is not exactly linearizable. This report presents a constructive technique for designing nonlinear controls for such systems. The simulations indicate that such nonlinear control laws can improve system performance, particularly trajectory tracking.

There are still many open issues to be resolved in constructing controllers for systems such as the acrobot. Due to the freedom in choosing the system approximation used to construct the control law, the performance of the overall method depends on the skill of the engineer in choosing a good approximation. Understanding how the choice of a given approximation affects the controller performance would be of great benefit in improving the results presented here. Unfortunately, there are currently very few tools in this area of approximation theory. Our own experiments with the acrobot indicate that intuition in this area is often misleading.

Another concern is the effect of the system approximation on the size of the region of stability. As mentioned in the introduction, for the acrobot it is desirable to make the regions of stability for a controller as large as possible in order to simplify the task of moving from the rest configuration to an inverted equilibrium point. But as the simulations of this section show (in particular, see Figure 10), the nonlinear controllers constructed here result in a small *decrease* in the region of attraction, at least in the slice of the state space presented. Once again, tools for analyzing the regions of attraction for a nonlinear system are not well developed.

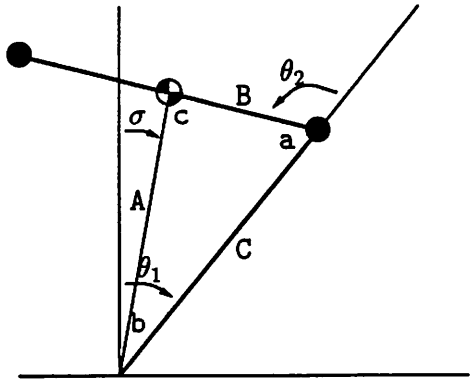

Figure 14: Acrobot center of mass geometry. The center of mass is located on the line between the first and second link.

A Balancing coordinates for the Acrobot

In this appendix we derive the equations for the angle of the center of mass of the acrobot as a function of the joint angles. Figure 14 shows the geometry of the problem. The relationship between the center of mass, σ, and the joint angles is

$$\begin{aligned} \sigma &= \theta_1 + b \\ \theta_1 &= \sigma - b \end{aligned} \tag{19}$$

where b is a function only of θ_2. The following identities hold for the triangle:

$$\begin{aligned} a &= \pi - \theta_2 \\ B &= \frac{m_2}{m_1+m_2} l_2 \\ C &= l_1 \end{aligned}$$

To calculate b given θ_2 we appeal first to the law of sines:

$$\frac{A}{\sin a} = \frac{B}{\sin b} = \frac{C}{\sin c} \quad \Rightarrow \quad \sin b = \frac{B \sin a}{A}$$

A can be determined by using the law of cosines:

$$A^2 = B^2 + C^2 - 2BC \cos a$$

Putting all of the equations together yields the desired formula

$$b = \sin^{-1} \frac{m_2 l_2 \sin \theta_2}{\sqrt{l_2^2 m_2^2 + (m_1+m_2)^2 l_1^2 + 2 l_1 l_2 m_2 (m_1+m_2) \cos \theta_2}} \tag{20}$$

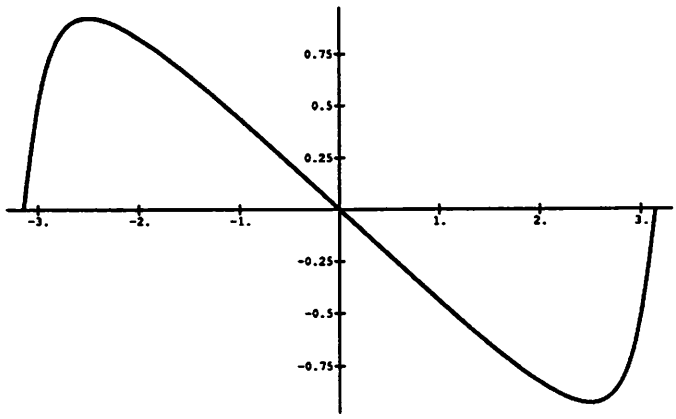

Figure 15: b versus θ_2 for the UC Berkeley Acrobot.

Using this equation and equation (19) gives the diffeomorphism between (θ_1, θ_2) and (θ_2, σ).

A plot of b as a function of θ_2 for the UC Berkeley Acrobot is show in Figure 15. It is clear from this picture that for $\theta_2 < \pi/2$, b is well approximated by a simple linear function. The slope of equation (20) at the origin is given by

$$\frac{l_2 m_2}{l_1 m_1 + (l_1 + l_2) m_2}.$$

B Mathematica listings

This appendix contains listings for the Mathematica code used to analyze the acrobot. The following files are included

acrobot.m	dynamic equations for the acrobot
approximate.m	approximate linearization
attraction.m	calculate region of attraction for control laws
balance.m	change of coordinates to "balancing" coordinates
compare.m	generate controller comparisons
equilibrium.m	parameterization of the equilibrium manifold
exact.m	check involutivity conditions for feedback linearization
linear.m	linear controller definition
linearize.m	linearization calculations
output.m	construct an artificial output function
schedule.m	gain-scheduled controller
stability.m	stability simulations
tracking.m	tracking simulations

Simulations for the acrobot were performed using a Mathematica-based simulation program, `Simulate.m`. Listings for `Simulate.m` are not included here; for further information, contact the authors.

```
(* acrobot.m - dynamic equations for Acrobot *)
Needs["Jac`"];
<<Trigonometry.m                (* trigonometric simplification *)

(* Unprotect the C[] function for Mathematica 1.2 *)
Unprotect[C];  C = .;

(* Short function for makeing lists of rules *)
SetAttributes[listRule, Listable];
listRule[lhs_,rhs_] := Rule[lhs, rhs];

(*
 * Dynamics
 *
 * We use the following definitions in deriving the equations of motion:
 *
 *      link 1: angle = theta1 (from vertical); length = l1; mass = m1
 *      link 2: angle = theta2 (from link 1); length = l2; mass = m2
 *
 *      a = m1 l1^2 + m2 l1^2
 *      b = m2 l2^2
 *      c = m2 l1 l2
 *      d = g (m1 l1 + m2 l1)
 *      e = g m2 l2
 *
 * Rather than let mathematica spin its wheels on Lagrange's equtaions,
 * we calculate them out by hand and write the dynamics in the same form
 * as we usually use for robot dynamics (i.e. use mass and coriolis terms)
 *)

SetAttributes[{a,b,c,d,e}, Constant]
SetAttributes[{m1,m2,l1,l2,gr}, Constant]

(* define a function to substitute original parameters into an expression *)
phys[x_] := x /. { a -> m1 l1^2 + m2 l1^2, b -> m2 l2^2,
                   c -> m2 l1 l2, d -> gr (m1 l1 + m2 l1), e -> gr m2 l2 }

(* define another function which actually uses *balanced* acrobot numbers *)
balance[x_] := phys[x] /. {l1->1/2, l2->1, m1->8, m2->8, gr->10}
real[x_] := phys[x] /. {l1->1/2, l2->3/4, m1->7, m2->8, gr->98/100}

(* the default set of parameters is the balanced set *)
num[x_] := balance[x];
Num[x_] := N[num[x]];

(* Shorthand vectors *)
q := {x1, x2}
w := {x3, x4}
x := {x1, x2, x3, x4}

(* mass matrix and coriolis matrix *)
(* use relative coords for second link *)
M[q_] := {{a + b + 2 c Cos[q[[2]]], b + c Cos[q[[2]]]}, {b + c Cos[q[[2]]], b}}
C[q_, w_] = {
  {-c Sin[q[[2]]] w[[2]], -c Sin[q[[2]]] (w[[1]] + w[[2]])},
  {c Sin[q[[2]]] w[[1]], 0}
}

(* Gravity and friction *)
v[q_] := d Cos[q[[1]]] + e Cos[q[[1]] + q[[2]]]
G[q_] := Jac[v[q], q]

(* Nonlinear vector fields *)
f[q_, w_] := Join[w, -Inverse[M[q]] . (C[q,w].w + G[q])];
f[{x1_, x2_, x3_, x4_}] = f[{x1, x2}, {x3, x4}];

g[q_, w_] := Join[{0,0}, Inverse[M[q]].{0, 1}];
g[{x1_, x2_, x3_, x4_}] = g[{x1, x2}, {x3, x4}];
```

```
(*
 * Approximate linearization
 *
 * Now we generate the change of coordinates which linearizes a
 * system with approximately the vector fields that we have.
 *
 *)

Print["calculating linearizing transformation"];

(* New coordinates after this transformation *)
z = {z1, z2, z3, z4};

(* Rule to kill off higher order terms *)
horule = {y2 y3 -> 0, y2 y4 -> 0, y3 y4 -> 0, y2^2->0, y3^2->0, y4^2->0};

phi1 = he;
Print["phi1: ", LeafCount[phi1], " terms"];

phi2 = Together[Expand[LieD[fy, phi1, y]] /. horule];
Print["phi2: ", LeafCount[phi2], " terms"];

phi3 = Together[Expand[LieD[fy, phi2, y]] /. horule];
Print["phi3: ", LeafCount[phi3], " terms"];

phi4 = Together[Expand[LieD[fy, phi3, y]] /. horule];
Print["phi4: ", LeafCount[phi4], " terms"];

(* Calculate the feedback law for stabilization *)
ay = Together[Expand[LieD[gy, phi4, y]] /. horule];
Print["ay: ", LeafCount[ay], " terms"];

by = Together[Expand[LieD[fy, phi4, y]] /. horule];
Print["by: ", LeafCount[by], " terms"];

Print["building nonlinear simulation"];

appreduce[expr_] := N[expr /.
  {beta0[y1] -> beta0v, beta1[y1] -> beta1v,
   beta2[y1] -> beta2v, beta3[y1] -> beta3v}, 16]

BuildSystem[nonlinear,
    States[x];
    Derivs[{dx1, dx2, dx3, dx4}];
    Inputs[{u}];
    Outputs[z];
    Outputs[{xi1d, xi2d, xi3d, xi4d}];

    Local[y];                   (* balance and linearizing coordinates *)
    Local[{v}];                 (* linearized input *)
    Local[{t1,t2}];             (* coriolis, gravity and input terms *)
    Local[{h1,h2,h3,h4,h5}];    (* the output function and derivatives *)
    Local[{beta0v, beta1v, beta2v, beta3v}];

    (* Default gain parameters *)
    Params[{K1=150.0625, K2=171.5, K3=73.5, K4=14}];
    Params[{amplitude=0, offset=0, omega=1}];
    Params[{type=0}];           (* input waveform *)
    Params[{alpha1=0.4397}];    (* slope of the eq mfd *)
    Params[{alpha3=0.00108799}];
    Params[{alpha5=-0.0018839}];
    Include["acrotraj.c"];      (* trajectory generator *)

    (* Turn off the simulation if things go crazy *)
    InlineC["if(fabs(x1) > 3 || fabs(x2) > 3) exit(3);"];

    (* Start by converting the state to balancing coordinates *)
    y = num[phiB[x] /. {beta0[ex_] -> beta[ex], beta1[ex_] -> dbeta[ex]}];

    (* Figure out the equilibrium mfd (beta) values *)
    beta0v = alpha1 * y1 + alpha3 * Power[y1,3] + alpha5 * Power[y1,5];
    beta1v = alpha1 + 3 * alpha3 * y1*y1 + 5 * alpha5 * Power[y1,4];
    beta2v = 6 * alpha3 * y1 + 20 * alpha5 * Power[y1,3];
    beta3v = 6 * alpha3 + 60 * alpha5 * Power[y1,2];

    (* Now got to linearizing coordinates *)
    z = appreduce[{phi1, phi2, phi3, phi4}];

    (* Trajectory update *)
    InlineC["acroTrajectory((int) type, t, omega, amplitude,
        offset, &h1,&h2,&h3,&h4,&h5);"];

    (* apply the control law *)
    v = h5 - {K1, K2, K3, K4} . (z - {h1, h2, h3, h4});
    u = appreduce[(-by + v) / ay];

    (* Acrobot dynamics - using the equations from above *)
    {t1, t2} = Num[-C[q, w].w - G[q] + {0, u}];
    {dx1, dx2} = w;
    {dx3, dx4} = Num[Inverse[M[q]] . {t1, t2}];

    (* Store the desired and actual trajectories in transformed coords *)
    {xi1d, xi2d, xi3d, xi4d} = {h1, h2, h3, h4};
]
```

```
(*
 * attraction.m - figure out the region of attraction by numerical simulation
 *
 * Richard M. Murray
 * June 13, 1990
 *
 *)

VectorCheck[system_, interval_, vector_, min_, max_, eps_] :=
  Block[
    {scale, good = min, bad = max, result},

     For[scale = min, bad-good > eps, scale = good + (bad-good)/2,
       (* Simulate the system and see if it is stable *)
       result = Simu[system, interval, Initial->N[(scale vector)]];

       (* Reset good or bad depending on the result *)
       If [SameQ[result, {}], bad = scale, good = scale];
    ];
    good
  ]

VectorMax[system_, theta_] :=
  Block[
    {x = Cos[theta], y = Sin[theta]},
    Print["theta = ", theta];
    {x, y} * VectorCheck[system, {0,10}, {x,y,0,0}, 0.0, 1.0, 0.01]
  ]

Parm[linear, amplitude->0];
Parm[nonlinear, amplitude->0];
LNdata = Table[VectorMax[linear, (i-1) 2 Pi / 16], {i, 1, 17}];
NLdata = Table[VectorMax[nonlinear, (i-1) 2 Pi / 16], {i, 1, 17}];

graph = Show[
  Graphics[
    {Line[NLdata], Dashing[{.02,.02}], Line[LNdata],
     Text["Nonlinear", Scaled[{0.1,0.035}], {-1,0}],
     Dashing[{1}], Line[{Scaled[{0,0.035}], Scaled[{0.075,0.035}]}],
     Text["Linear",    Scaled[{0.1,0.070}], {-1,0}],
     Dashing[{.02, .02}], Line[{Scaled[{0,.070}], Scaled[{0.075,.070}]}]},
     Axes->Automatic, AxesLabel->{"th1", "th2"}, PlotRange->All
  ]
];

Display["attraction.mps", graph];
```

```
(*
 * balance.m - Change of coordinates to "balancing" coordinates
 *
 * We now construct a change of coordinates y = phiB[x] so that the
 * equlibrium manifold is just {y1, 0, 0, 0}.
 *
 * This construction assumes we are using idealized coordinates.  This
 * is also a good approximation for non-ideal coordinates (see ERL appendix)
 *
 *)
Print["calculating balancing coordinates"];

y := {y1, y2, y3, y4}
phiB[x_List] := {x[[2]], x[[1]] + l2 m2 x[[2]] / (l1 m1 + (l1 + l2) m2),
                 x[[4]], x[[3]] + l2 m2 x[[4]] / (l1 m1 + (l1 + l2) m2)}
ihpB[y_List] := {y[[2]] - l2 m2 y[[1]] / (l1 m1 + (l1 + l2) m2), y[[1]],
                 y[[4]] - l2 m2 y[[3]] / (l1 m1 + (l1 + l2) m2), y[[3]]}

(* Push the vector fields through the change of coordinates *)
(* This is slightly simplified since the coordinate change is linear *)
fBy = Together[ Jac[phiB[x], x] . f[ihpB[y]] ];
gBy = Together[ Jac[phiB[x], x] . g[ihpB[y]] ];

(* Figure out what the equilibrium input must be in order to balance *)
(* See equilibrium.m for comments *)
eqrule = listRule[Last[y], 0];
ue = {u -> num[ -m2 l2 gr Sin[x1+x2] /. listRule[x, ihpB[y]] /. eqrule]};
```

```
(*
 * compare.m - generate plots for the controller comparisons chapter
 *
 * Richard M. Murray
 * August 31, 1990
 *
 * All simulations generated by this file use the *real* values of acrobot
 * instead of the ideal values.  (RMM 4/2/91)
 *)

(* Read in the basic description of acrobot *)
<<acrobot.m

(* Express vector fields in equilibrium coordinates *)
(* Use special routine optimized for balanced parameters *)
<<balance.m

(* Use a linear approximation to the equlibrium manifold *)
(*
Clear[alpha1]                   (* alpha1 = eq mfd slope (keep symbolic) *)
Print["using approximate balancing coordinates"]
<< appbalance.m
<< appoutput.m                  (* read a save output function *)
*)

(* Generate the output function used by all controllers *)
<< output.m
he = FindOutput[fBy, gBy, ue, y]

(* Desired trajectory; set amplitude = 0 for setpoint tracking *)
desired = amplitude Sin[omega t] + offset;
SetAttributes[{amplitude,offset, omega}, Constant];

(* Build the individual systems *)
<< Simulate.m
<< linear.m
<< schedule.m
<< approximate.m

Print["Using lsoda integrator"]
Map[SetOptions[#, Method->lsoda]&, {linear,schedule,nonlinear}]

(* Mathematica generated plots *)
(* Run this manually; it takes a while to finish *)
(* << linearize.m *)

(* Now run the simulations for each part of the chapter *)
(* These should be run manually since they take a while *)
(* << stability.m *)                    (* set point stability *)
(* << attraction.m *)                   (* region of attraction *)
(* << tracking.m *)                     (* tracking comparisons *)
(* << poles.m *)                        (* effect of moving poles *)
```

```
(*
 * Change of coordinates to "balancing" coordinates
 *
 * We now construct a change of coordinates y = phiB[x] so that the
 * equlibrium manifold is just {y1, 0, 0, 0}.  This construction holds
 * for general parameter values.
 *
 *)
Print["calculating balancing coordinates"];

y := {y1, y2, y3, y4}
phiB[x_List] :=
  Block[
    (* Use the law of cosines to get the inner angle *)
    {th1, th2, beta},
    beta = -ArcSin[ Sin[th2] /
      Sqrt[1 + (m1+m2)^2 l1^2 / (l2 m2)^2 + 2(m1+m2)l1 Cos[th2] / (l2 m2)] ];

    (* Now figure out the coordinates + velocities *)
    {x[[2]], th1-beta, x[[4]], Jac[th1-beta, {th1, th2}] . x{{{3,4}}]} /.
    {th1->x[[1]], th2->x[[2]]}
]

ihpB[y_List] :=
  Block[
    (* Use the law of cosines to get the inner angle *)
    {sig, th2, beta},
    beta = -ArcSin[ Sin[th2] /
      Sqrt[1 + (m1+m2)^2 l1^2 / (l2 m2)^2 + 2(m1+m2)l1 Cos[th2] / (l2 m2)] ];

    (* Now figure out the coordinates + velocities *)
    {y[[2]]+beta, y[[1]], Jac[sig+beta, {th2, sig}] . y{{{3,4}]], y[[3]]} /.
    {th2->y[[1]], sig->y[[2]]}
]

(* Push the vector fields through the change of coordinates *)
acroSimplify[e_] := Together[Expand[num[e]]]
fBy = acroSimplify[(Jac[phiB[x], x] . f[x]) /. listRule[x, ihpB[y]]];
gBy = acroSimplify[(Jac[phiB[x], x] . g[x]) /. listRule[x, ihpB[y]]];

(*
 * Figure out what the equilibrium input must be in order to balance
 *
 * This used to be a linear equation, but the general acrobot doesn't
 * support that.  Fortunately, we can write down the solution in closed
 * form in the original set of coordinates.
 *
 * Old code:
 *    solns = Solve[(fy + gy u //. eqrule) == 0, u];
 *    If[Length[solns] != 1,
 *      Print["Can't solve equations for equilibrium input\n"]; ue = 0,
 *     ue = solns[[1]]];
 *
 *)
eqrule = listRule[Last[y], 0];
ue = {u -> num[ -m2 l2 gr Sin[x1+x2] /. listRule[x, ihpB[y]] /. eqrule]};
```

```
(*
 * exact.m - check involutivity conditions for acrobot
 *
 * Richard M. Murray
 * August 30, 1989
 *
 *)

Needs["Jac`"];
Print["calculating brackets"]

(* We have to use numerical values here to make the computation work *)
f0 = Factor[Together[num[f[x]]]];            Print["f: ", LeafCount[f0]];
c1 = Factor[Together[num[g[x]]]];            Print["g: ", LeafCount[c1]];
c2 = Factor[Together[Lie[f0,c1,x]]];    Print["[f, g]: ", LeafCount[c2]];
c3 = Factor[Together[Lie[f0,c2,x]]];    Print["[f^2, g]: ", LeafCount[c3]];

Print["checking involutivity"]
c4 = Factor[Together[Lie[c1, c2, x]]];
d1 = Together[Det[{c1, c2, c3, c4}]];   Print["d1 = ", d1];

(* This calculation takes a while *)
c5 = Factor[Together[Lie[c2, c3, x]]];
d2 = Together[Det[{c1, c2, c3, c5}]];        Print["d2 = ", d2];

c6 = Factor[Together[Lie[c1, c3, x]]];
d3 = Together[Det[{c1, c2, c3, c6}]];        Print["d3 = ", d3];
```

```
(*
 * Linear simulation
 *
 * Just use state static feedback to try to control the system
 *)

Print["building linear simulation"];

(* New coordinates after this transformation *)
z = {z1, z2, z3, z4};

A0 = num[Jac[f[x], x] /. {x1->0, x2->0, x3->0, x4->0}];
b0 = num[g[x]         /. {x1->0, x2->0, x3->0, x4->0}];
c0 = num[Jac[ he /. listRule[y, phiB[x]], x ] /.
        {x1->0, x2->0, x3->0, x4->0}] /. {alpha3->0, alpha5->0} /.
        {beta0[y1_] -> alpha0 y1, beta1[y1_] -> alpha0, beta2[y1_] ->0} /.
        {num[alpha0->l2 m2 / (l1 m1 + (l1 + l2) m2)]}

(* Define linear change of coordinates (=> we can use the same gains) *)
BuildSystem[linear,
    States[x];
    Derivs[{dx1, dx2, dx3, dx4}];
    Inputs[{u}];
    Outputs[z];
    Outputs[{xi1d, xi2d, xi3d, xi4d}];

    Local[y];                   (* balance and linearizing coordinates *)
    Local[{v}];                 (* linearized input *)
    Local[{t1,t2}];             (* coriolis, gravity and input terms *)
    Local[{h1,h2,h3,h4,h5}];    (* the output function and derivatives *)

    Params[{K1=150.0625, K2=171.5, K3=73.5, K4=14}];
    Params[{amplitude=0, offset=0, omega=1}];
    Params[{type=0}];           (* input waveform *)
    Include["acrotraj.c"];      (* trajectory generator *)

    (* Turn off the simulation if things go crazy *)
    (* InlineC["printf(\"x1 = %g, x2 = %g\n\", x1, x2);"]; *)
    InlineC["if(fabs(x1) > 3 || fabs(x2) > 3) exit(3);"];

    (* Now got to linearizing coordinates *)
    z = {c0.x, c0.A0.x, c0.A0.A0.x, c0.A0.A0.A0.x};

    (* Trajectory update *)
    InlineC["acroTrajectory((int) type, t, omega, amplitude,
        offset, &h1,&h2,&h3,&h4,&h5);"];

    (* apply the control law *)
    v = h5 - {K1, K2, K3, K4} . (z - {h1, h2, h3, h4});
    u = (-c0.A0.A0.A0.A0.x + v) / (c0.A0.A0.A0.b0);

    (* Acrobot dynamics - using the equations from above *)
    {t1, t2} = Num[-C[q, w].w - G[q] + {0, u}];
    {dx1, dx2} = w;
    {dx3, dx4} = Num[Inverse[M[q]] . {t1, t2}];

    (* Store the desired and actual trajectories in transformed coords *)
    {xi1d, xi2d, xi3d, xi4d} = {h1, h2, h3, h4};
]
```

```
(*
 * linearize.m - check out the linearization of acrobot
 *
 * Richard M. Murray
 * June 2, 1990
 *
 *)

(* Figure figure out the linearization about the vertical equilibrium point *)
A0 = num[Jac[f[x], x] /. {x1->0, x2->0, x3->0, x4->0}];
b0 = num[g[x]         /. {x1->0, x2->0, x3->0, x4->0}];

(* Controllable ??*)
Print["Linear controllability = ", Det[{b0, A0.b0, A0.A0.b0, A0.A0.A0.b0}]];

(* Now figure out the points at which we loose controllability *)
uA = real[Jac[f[x], x] /. {x1->-x2/2, x3->0, x4->0}];
uB = real[g[x]         /. {x1->-x2/2, x3->0, x4->0}];

bA = balance[Jac[f[x], x] /. {x1->-x2/2, x3->0, x4->0}];
bB = balance[g[x]         /. {x1->-x2/2, x3->0, x4->0}];

balDet[v_] :=
  Det[Map[N[# /. x2->v]&, {bB, bA.bB, bA.bA.bB, bA.bA.bA.bB}]];
balSigma[v_] :=
  SingularValues[
    Map[N[# /. x2->v]&, {bB, bA.bB, bA.bA.bB, bA.bA.bA.bB}]
  ][[2]];

ucbDet[v_] :=
  Det[Map[N[# /. x2->v]&, {uB, uA.uB, uA.uA.uB, uA.uA.uA.uB}]];
ucbSigma[v_] :=
  SingularValues[N[{uB, uA.uB, uA.uA.uB, uA.uA.uA.uB} /. x2->v]][[2]];

ucbGraph = Plot[ucbDet[x2], {x2, -Pi, Pi}]
Display["ucbctrl.mps", ucbGraph]

balGraph = Plot[balDet[x2], {x2, -Pi, Pi}]
Display["balctrl.mps", balGraph]
```

```
(*
 * Determination of the output function
 *
 * We now calculate the output function by using the linearization along
 * the equilibrium manifold (y2 = y3 = y4 = 0).  This makes use of
 * a special set of parameters to simplify calculations, so from here
 * on out, all of the calculations are numerical.
 *
 *)

FindOutput[fBy_, gBy_, ue_, y_] :=
  Block[
    {Ae, be, ce, W, i},
    Print["determining output function"];

    (* Get the vector fields and evaluate them *)
    (*! These are required by functions outside of this one !*)
    fy = num[fBy];
    gy = num[gBy];

    (* Make a rule to evaluate an expression at an equilibrium point *)
    eqrule = listRule[Rest[y], 0];

    (* Now figure out what the linearization is along the eq manifold *)
    Print["  calculating linearization along eq manifold"];
    Ae = Jac[fy + gy u, y] /. ue //. eqrule;
    be = gy //. eqrule;

    (*
     * Originally we calculated the output function by using the controllable
     * canonical form of the linearized system.  Now we know better, so we
     * just look at the null space of the controllability matrix
     *
     *)
    Print["  null space calculation"];
    W = {be};
    For[i = 2, i < Length[be], ++i, W = Join[W, {Ae.Last[W]}]];
    ce = Together[ NullSpace[W][[1]] ];

    (* Return the normalized output function *)
    Together[ ce/ce[[1]] ] . y
]
```

```
(*
 * Gain scheduling
 *
 * Now we generate the change of coordinates which linearizes a
 * system with approximately the vector fields that we have.
 *
 *)

Print["calculating linearizing transformation"];

(* New coordinates after this transformation *)
z = {z1, z2, z3, z4};

(* Rule to kill off higher order terms *)
horule = {y2 y3 -> 0, y2 y4 -> 0, y3 y4 -> 0, y2^2->0, y3^2->0, y4^2->0};

normal[expr_SeriesData] := Normal[expr];
normal[expr_] := expr;

expand[expr_] := Expand[normal[Series[expr, {y2,0,1}]]]
reduce[expr_] := Together[expand[expr] /.
  horule //. {Hold[Normal[a_]]->a}];

(* Build the transformation; use Taylor series to get linear truncation *)
phi1 = reduce[he];
Print["phi1: ", LeafCount[phi1], " terms"];

phi2 = reduce[LieD[fy, phi1, y]]
Print["phi2: ", LeafCount[phi2], " terms"];

phi3 = reduce[LieD[fy, phi2, y]]
Print["phi3: ", LeafCount[phi3], " terms"];

phi4 = reduce[LieD[fy, phi3, y]]
Print["phi4: ", LeafCount[phi4], " terms"];

(* Calculate the feedback law for stabilization *)
(* Truncate the linear terms *)
ay = reduce[LieD[gy, phi4, y]];
Print["ay: ", LeafCount[ay], " terms"];

by = reduce[LieD[fy, phi4, y]];
Print["by: ", LeafCount[by], " terms"];

Print["building schedule simulation"];

BuildSystem[schedule,
    States[x];
    Derivs[{dx1, dx2, dx3, dx4}];
    Inputs[{u}];
    Outputs[z];
    Outputs[{xi1d, xi2d, xi3d, xi4d}];

    Local[y];                   (* balance and linearizing coordinates *)
    Local[{v}];                 (* linearized input *)
    Local[{t1,t2}];             (* coriolis, gravity and input terms *)
    Local[{h1,h2,h3,h4,h5}];    (* the output function and derivatives *)
    Local[{beta0v, beta1v, beta2v, beta3v}];

    (* Default gain parameters *)
    Params[{K1=150.0625, K2=171.5, K3=73.5, K4=14}];
    Params[{amplitude=0, offset=0, omega=1}];
    Params[{type=0}];           (* input waveform *)
    Params[{alpha1=0.4397}];    (* slope of the eq mfd *)
    Params[{alpha3=0.00108799}];
    Params[{alpha5=-0.0018839}];

    Include["acrotraj.c"];      (* trajectory generator *)

    (* Turn off the simulation if things go crazy *)
    InlineC["if(fabs(x1) > 3 || fabs(x2) > 3) exit(3);"];

    (* Start by converting the state to balancing coordinates *)
    y = num[phiB[x] /. {beta0[ex_] -> beta[ex], beta1[ex_] -> dbeta[ex]}];

    (* Figure out the equilibrium mfd (beta) values *)
    beta0v = alpha1 * y1 + alpha3 * Power[y1,3] + alpha5 * Power[y1,5];
    beta1v = alpha1 + 3 * alpha3 * y1*y1 + 5 * alpha5 * Power[y1,4];
    beta2v = 6 * alpha3 * y1 + 20 * alpha5 * Power[y1,3];
    beta3v = 6 * alpha3 + 60 * alpha5 * Power[y1,2];

    (* Now got to linearizing coordinates *)
    z = {phi1, phi2, phi3, phi4};

    (* Trajectory update *)
    InlineC["acroTrajectory((int) type, t, omega, amplitude,
        offset, &h1,&h2,&h3,&h4,&h5);"];

    (* apply the control law *)
    v = h5 - {K1, K2, K3, K4} . (z - {h1, h2, h3, h4});
    u = (-by + v) / ay;

    (* Acrobot dynamics - using the equations from above *)
    {t1, t2} = Num[-C[q, w].w - G[q] + {0, u}];
    {dx1, dx2} = w;
    {dx3, dx4} = Num[Inverse[M[q]] . {t1, t2}];

    (* Store the desired and actual trajectories in transformed coords *)
    {xi1d, xi2d, xi3d, xi4d} = {h1, h2, h3, h4};
]
```

```
(* Tracking simulations *)
Parm[schedule,    amplitude->1, offset->0, omega->1, type->0]
Simu[schedule,    {0, 20, .02}, Output->"tracking.sch", Initial->{0, 0, 0, 0}]

Parm[nonlinear, amplitude->1, offset->0, omega->1, type->0]
Simu[nonlinear, {0, 20, .02}, Output->"tracking.nlc", Initial->{0, 0, 0, 0}]

Parm[linear,    amplitude->1, offset->0, omega->1, type->0]
Simu[linear,    {0, 20, .02}, Output->"tracking.lin", Initial->{0, 0, 0, 0}]
```

```
(* Simulate the system with some various starting points *)
Parm[nonlinear, K1->150.0625, K2->171.5, K3->73.5, K4->14]
Parm[nonlinear, amplitude->0, offset->0]
(*
Simu[nonlinear, {0, 10}, Output->"nonlinear.1", Initial->{0, 0.01, 0, 0}]
Simu[nonlinear, {0, 10}, Output->"nonlinear.2", Initial->{0.01, 0, 0, 0}]
Simu[nonlinear, {0, 10}, Output->"nonlinear.3", Initial->{-0.02, 0.03, 0, 0}]
Simu[nonlinear, {0, 10}, Output->"nonlinear.4", Initial->{0.1, 0, 0, 0}]
*)
Simu[nonlinear, {0, 5, .02}, Output->"stability.non", Initial->{0, 0.2, 0, 0}]

(* Simulate the system with some various starting points *)
Parm[schedule, K1->150.0625, K2->171.5, K3->73.5, K4->14]
Parm[schedule,    amplitude->0, offset->0]
(*
Simu[schedule, {0, 10}, Output->"schedule.1", Initial->{0, 0.01, 0, 0}]
Simu[schedule, {0, 10}, Output->"schedule.2", Initial->{0.01, 0, 0, 0}]
Simu[schedule, {0, 10}, Output->"schedule.3", Initial->{-0.02, 0.03, 0, 0}]
Simu[schedule, {0, 10}, Output->"schedule.4", Initial->{0.1, 0, 0, 0}]
*)
Simu[schedule, {0, 5, .02}, Output->"stability.sch", Initial->{0, 0.2, 0, 0}]

(* Simulate the system with some various starting points *)
Parm[linear, K1->150.0625, K2->171.5, K3->73.5, K4->14]
Parm[linear,    amplitude->0, offset->0]
(*
Simu[linear, {0, 10}, Output->"linear.1", Initial->{0, 0.01, 0, 0}]
Simu[linear, {0, 10}, Output->"linear.2", Initial->{0.01, 0, 0, 0}]
Simu[linear, {0, 10}, Output->"linear.3", Initial->{-0.02, 0.03, 0, 0}]
Simu[linear, {0, 10}, Output->"linear.4", Initial->{0.1, 0, 0, 0}]
*)
Simu[linear, {0, 5, .02}, Output->"stability.lin", Initial->{0, 0.2, 0, 0}]

Null
```

[13] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley, 1989.

References

[1] W. T. Baumann and W. J. Rugh. Feedback control of nonlinear systems by extended linearization. *IEEE AC Transactions*, 31:40–46, 1986.

[2] C. I. Byrnes and A. Isidori. Local stabilization of minimum-phase nonlinear systems. *Systems and Control Letters*, 11:9–17, 1988.

[3] J. Hauser, S. Sastry, and P. Kokotović. Nonlinear control via approximate input-output linearization, the ball and beam example. Technical Report ERL, Department of EECS, University of California, Berkeley, 1989. To appear in *IEEE Transactions on Automatic Control*, 1991.

[4] John Hauser. Nonlinear control via uniform nonlinear system approximation. In *IEEE Control and Decision Conference*, 1990. To appear in *Systems and Control Letters*, 1991.

[5] L. R. Hunt, R. Su, and G. Meyer. Global transformations of nonlinear systems. *IEEE AC Transactions*, AC-28(1):24–31, 1983.

[6] A. Isidori. *Nonlinear Control Systems.* Springer-Verlag, 2nd edition, 1989.

[7] B. Jakubczyk and W. Respondek. On linearization of control systems. *Bulletin de L'Academie Polonaise des Sciences, Série des sciences mathématiques*, XXVIII:517–522, 1980.

[8] A. J. Krener. Approximate linearization by state feedback and coordinate change. *Systems and Control Letters*, 5:181–185, 1984.

[9] A. J. Krener, S. Karahan, M. Hubbard, and R. Frezza. Higher order linear approximations to nonlinear control systems. In *IEEE Control and Decision Conference*, pages 519–523, 1987.

[10] C. Reboulet and C. Champetier. A new method for linearizing nonlinear systems: the pseudolinearization. *International Journal of Control*, 40:631–638, 1984.

[11] M. W. Spong and M. Vidyasagar. *Dynamics and Control of Robot Manipulators.* John Wiley, 1989.

[12] J. Wang and W. J. Rugh. On the pseudo-linearization problem for nonlinear systems. *Systems and Control Letters*, 12:161–167, 1989.