In H. Heuer & S. Keele, (Eds.), *Handbook of Perception and Action: Motor Skills*. New York: Academic Press, 1996.

Computational aspects of motor control and motor learning*

Michael I. Jordan
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology

*I want to thank Elliot Saltzman, Steven Keele, and Herbert Heuer for helpful comments on the manuscript. Preparation of this paper was supported in part by a grant from ATR Auditory and Visual Perception Research Laboratories, by a grant from Siemens Corporation, by a grant from the Human Frontier Science Program, by a grant from the McDonnell-Pew Foundation, and by grant N00014-90-J-1942 awarded by the Office of Naval Research. Michael Jordan is an NSF Presidential Young Investigator.

This chapter provides a basic introduction to various of the computational issues that arise in the study of motor control and motor learning. A broad set of topics is discussed, including feedback control, feedforward control, the problem of delay, observers, learning algorithms, motor learning, and reference models. The goal of the chapter is to provide a unified discussion of these topics, emphasizing the complementary roles that they play in complex control systems. The choice of topics is motivated by their relevance to problems in motor control and motor learning; however, the chapter is not intended to be a review of specific models. Rather we emphasize basic theoretical issues with broad applicability.

Many of the ideas described here are developed more fully in standard textbooks in modern systems theory, particularly textbooks on discrete-time systems (Åström & Wittenmark, 1984), adaptive signal processing (Widrow & Stearns, 1985), and adaptive control systems (Goodwin & Sin, 1984; Åström & Wittenmark, 1989). These texts assume a substantial background in control theory and signal processing, however, and many of the basic ideas that they describe can be developed in special cases with a minimum of mathematical formalism. There are also issues of substantial relevance to motor control that are not covered in these standard sources, particularly problems related to nonlinear systems and time delays. As we shall see, consideration of these problems leads naturally to a focus on the notion of an "internal model" of a controlled system. Much of the discussion in the chapter will be devoted to characterizing the various types of internal models and describing their role in complex control systems.

In the next several sections, we develop some of the basic ideas in the control of dynamical systems, distinguishing between *feedback* control and *feedforward* control. In general, controlling a system involves finding an input to the system that will cause a desired behavior at its output. Intuitively, finding an *input* that will produce a desired *output* would seem to require a notion of "inverting" the process that leads from inputs to outputs; that is, controlling a dynamical system would seem to involve the notion of "inverting" the dynamical system. As we will see, this notion can be made precise and made to serve as a useful unifying principle for understanding control systems. Indeed, feedback control and feedforward control can both be understood as techniques for inverting a dynamical system. Before developing these ideas we first discuss some mathematical representations for dynamical systems.

Dynamical Systems

A fundamental fact about many systems is that knowledge of only the input to the system at a given time is not sufficient to predict its output. For example, to predict the flight of a ping-pong ball, it is not enough to know how it was struck by the paddle but it is also necessary to know its velocity and spin prior to being struck. Similarly, to predict the effects of applying a torque around the knee joint one must know the configuration and motion of the body. In general, to predict the effect of the input to a dynamical system, one must know the values of an additional set of variables known as *state variables.* Knowing the state of a system and its input at a given moment in time is sufficient to predict its state at the next moment in time.

In physical systems, the states of a system often have a natural physical interpretation. For example, knowing the position and velocity of a mass together with the force acting on the mass is sufficient to predict the position and velocity of the mass at the next instant of time. Thus position and velocity are the state variables and force is the input variable.

It is also common to specify an *output* of a dynamical system. Mathematically the output is simply a specified function of the state of the system. In many cases, the output has a physical interpretation as the set of measurements made by an external measuring device. In other cases, the choice of the output variables is dictated more by the goals of the modeler than by the existence of a measuring device. For example, the modeler of the ping-pong ball may be interested in tracking the kinetic and potential energy of the ball, perhaps as part of a theoretical effort to understand the ball's motion. In such a case the kinetic and potential energy of the ball, both of which are functions of the state, would be the output variables.

In general, a dynamical system can be characterized by a pair of equations: a *next-state equation* that expresses how the state changes as a function of the current state $\mathbf{x}[n]$ and the input $\mathbf{u}[n]$:

$$\mathbf{x}[n+1] = f(\mathbf{x}[n], \mathbf{u}[n]), \tag{1}$$

where n is the time step, and an *output equation* that specifies how the output $\mathbf{y}[n]$ is obtained from the current state:

$$\mathbf{y}[n] = g(\mathbf{x}[n]).^1 \tag{2}$$

[1] We use discrete time throughout the chapter, mainly for pedagogical reasons. In the following section an example is given of converting a continuous-time dynamical system to a corresponding discrete-time dynamical system.

The functions f and g are referred to as the next-state function and the output function, respectively. It is often useful to combine these equations and write a composite equation that describes how states and inputs map into outputs:

$$\mathbf{y}[n+1] = h(\mathbf{x}[n], \mathbf{u}[n]), \tag{3}$$

where h is the composition of f and g.

Many sensorimotor transformations are naturally expressed in terms of state space models. To model speech production, for example, one might choose the positions and velocities of the speech articulators as state variables and the muscular forces acting on the articulators as input variables. The next-state equation would characterize the motion of the articulators. A natural choice of output variables for speech would be a spectral representation of the speech signal, thus the output equation would model the acoustics of the vocal tract.

There is another representation for dynamical systems that does away with the notion of state in favor of a representation in terms of sequences of input vectors. Consider again the ping-pong example: the velocity and spin of the ping-pong ball at a given moment in time can be analyzed in terms of the way the ball was struck at the preceding time step, the time step before that, and so on. In general, a dynamical system can be treated as a transformation from an infinite sequence of input vectors to an output vector:

$$\mathbf{y}[n+1] = F(\mathbf{u}[n], \mathbf{u}[n-1], \mathbf{u}[n-2], \ldots). \tag{4}$$

This representation emphasizes the fact that a dynamical system can be treated as a mapping from an input sequence to an output sequence. The disadvantage of this representation is that the function F is generally much more complex that its counterparts f and g. In the remainder of the paper we assume that a dynamical system can be expressed in terms of a set of state variables and a pair of functions f and g.[2]

[2]It is worth noting that in many dynamical systems the influence of the input dies away over time, so that an input-output relation involving an infinite sequence of previous inputs (as in Equation 4) can often be *approximated* by a truncated relationship involving only the last K inputs:

$$\mathbf{y}[n+1] \approx \hat{F}(\mathbf{u}[n], \mathbf{u}[n-1], \ldots, \mathbf{u}[n-K+1]). \tag{5}$$

If we define the state variable $\mathbf{x}[n]$ as the sequence $\mathbf{u}[n-1], \mathbf{u}[n-2], \ldots, \mathbf{u}[n-K]$, then Equation 5 can be represented in terms of the state equations

$$\mathbf{x}[n+1] = f(\mathbf{x}[n], \mathbf{u}[n])$$

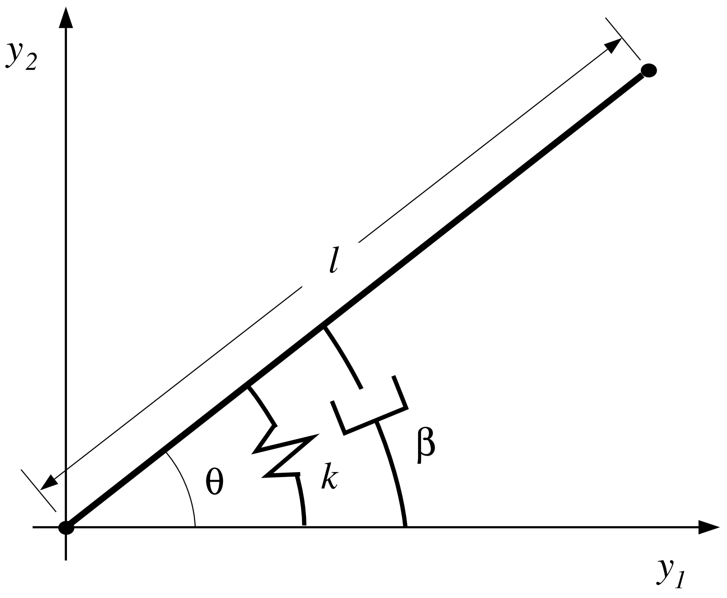

Figure 1: A one-link mechanical system. A link of length l is subject to torques from a linear spring with spring constant k and a linear damper with damping constant β.

Dynamics and kinematics

The term "dynamics" is used in a variety of ways in the literature on motor control. Two of the most common uses of the term derive from robotics (e.g., Hollerbach, 1982; Saltzman, 1979), and from dynamical systems theory (e.g., Turvey, Shaw, & Mace, 1978; Haken, Kelso, & Bunz, 1985). Let us review some of the relevant distinctions by way of an example.

Consider the one-link mechanical system shown in Figure 1. Newton's laws tell us that the angular acceleration of the link is proportional to the total torque acting on the link:

$$I\ddot{\theta} = -\beta\dot{\theta} - k(\theta - \theta_o), \tag{6}$$

where θ, $\dot{\theta}$, and $\ddot{\theta}$ are the angular position, velocity, and acceleration, respectively, I is the moment of inertia, β is the damping coefficient, k is the spring

and

$$\mathbf{y}[n] = g(\mathbf{x}[n]),$$

where g is equal to $\hat{F}$ and f simply shifts the current input $\mathbf{u}[n]$ into the state vector while shifting $\mathbf{u}[n-K]$ out. Thus truncated input-output representations can be easily converted to state variable representations.

constant, and θ_o is the equilibrium position of the spring. This equation can be approximated by a discrete-time equation of the form:[3]

$$\theta[t+2] = a_1\theta[t+1] + a_2\theta[t] + b\theta_o[t], \tag{7}$$

where $a_1 = 2 - h\beta/I$, $a_2 = h\beta/I - h^2k/I - 1$, $b = h^2k/I$, and h is the time step of the discrete-time approximation.

Let us suppose that movement of the link is achieved by controlled changes in the equilibrium position of the spring. Thus we define the control signal $u[t]$ to be the time-varying equilibrium position $\theta_o[t]$ (cf. Hogan, 1984). Let us also define two state variables:

$$\begin{aligned} x_1[t] &= \theta[t+1] \\ x_2[t] &= \theta[t]. \end{aligned}$$

Note that $x_2[t+1] = x_1[t]$. We combine this equation with Equation 7 to yield a single vector equation of the form:

$$\begin{pmatrix} x_1[t+1] \\ x_2[t+1] \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1[t] \\ x_2[t] \end{pmatrix} + \begin{pmatrix} b \\ 0 \end{pmatrix} u[t], \tag{8}$$

which is of the general form of a next-state equation (cf. Equation 1).

Let us also suppose that we want to describe the position of the tip of the link at each moment in time by a pair of Cartesian coordinates y_1 and y_2. Elementary trigonometry gives us:

$$\begin{pmatrix} y_1[t] \\ y_2[t] \end{pmatrix} = \begin{pmatrix} l\cos(x_2[t]) \\ l\sin(x_2[t]) \end{pmatrix}, \tag{9}$$

where l is the length of the link. This equation is an output equation of the form of Equation 2.

Let us now return to the terminological issues that we raised earlier. We have described a dynamical system in terms of a next-state equation (Equation 8) and an output equation (Equation 9). For a roboticist, the next-state equation in this example is the *dynamics* of the link and the output equation is the *kinematics* of the link. In general, a roboticist uses the term "dynamics" to refer to an equation that relates forces or torques to movement (e.g., accelerations). Such an equation generally corresponds to the next-state equation

[3] There are many ways to convert differential equations to difference equations. We have utilized a simple Euler approximation in which $\ddot{\theta}$ is replaced by $(\theta[t+2h]-2\theta[t+h]+\theta[t])/h^2$, and $\dot{\theta}$ is replaced by $(\theta[t+h]-\theta[t])/h$. For further discussion of discrete-time approximation of continuous-time dynamical systems, see Åström and Wittenmark (1984).

of a dynamical system. The term "kinematics" is used to refer to a transformation between coordinate systems (e.g., angular coordinates to Cartesian coordinates). This generally corresponds to the output equation of a dynamical system. To a dynamical systems theorist, on the other hand, the next-state equation and the output equation together constitute a "dynamical system." In this tradition, the term "dynamics" is used more broadly than in the mechanics tradition: Any mathematical model that specifies how the state of a system evolves specifies the "dynamics" of the system. No special reference need be made to forces or torques, nor, in many cases, to any notion of causality. Many useful dynamical systems models are simply descriptive models of the temporal evolution of an interrelated set of variables.

Forward and Inverse Models

The term "model" is also used with a variety of meanings in the motor control literature. Most commonly, a model is a formal system that a scientist uses to describe or explain a natural phenomenon. There are models of muscle dynamics, models of reaching behavior, or models of the cerebellum. Another usage of the term model is in the sense of "internal model." An internal model is a structure or process in the central nervous system that mimics the behavior of some other natural process. The organism may have an internal model of some aspect of the external world, an internal model of its own musculoskeletal dynamics, or an internal model of some other mental transformation. Note that it is not necessary for the internal structure of a model to correspond in any way to the internal structure of the process being modeled. For example, an internal model might predict the distance that a propelled object will travel, without integrating the equations of motion, either explicitly or implicitly. (The prediction could be based, for example, on extrapolation from previous observations of propelled objects). These two senses of "model" are also often merged; in particular when a scientist's model of a phenomenon (e.g., reaching) posits an internal model of a sensorimotor transformation (e.g., an internal kinematic model). This dual or composite sense of "model" captures the way in which we will often use the term in the remainder of the chapter. Thus a model of reaching may include a piece of formal machinery (generally a state space representation) that models the posited internal model.

Earlier sections introduced state space representations of dynamical systems. This mathematical framework requires a choice of variables to serve as inputs and a choice of variables to serve as outputs. For any particular dynamical system, the choice of variables is generally non-arbitrary and is conditioned by our understanding of the causal relationships involved. For example, in a dynamical model of angular motion it is natural to treat torque as an input variable and angular acceleration as an output variable. Models of motor control also tend to treat the relationships between variables in terms of a causal, directional flow. Certain variables are distinguished as motor variables and other variables are distinguished as (reafferent) sensory variables. In a dynamical model the motor variables are generally treated as inputs and the sensory variables are generally treated as outputs.

There are other transformations in motor control that are not generally conceived of in terms of causality, but which nonetheless are usefully thought of in terms of a directional flow. For example, the relationship between the joint coordinates of an arm and the spatial coordinates of the hand is not a causal relationship; however, it is still useful to treat spatial coordinates as being derived from joint coordinates. This is due to the *functional* relationship between joint coordinates and spatial coordinates. As is well known (e.g., Bernstein, 1967), the relationship between joint angles and spatial positions is many-to-one; that is, to any given spatial position of the hand there generally corresponds an infinite set of possible configurations of the joints. Thus the joint coordinates and the spatial coordinates have asymmetric roles in describing the geometry of the limb. This functional asymmetry parallels the asymmetry that arises from causal considerations and allows us to impose a directionality on sensorimotor transformations. We refer to the many-to-one, or causal, direction as a *forward* transformation, and the one-to-many, or anti-causal, direction as an *inverse* transformation.

The preceding considerations lead us to distinguish between forward models and inverse models of dynamical systems. Whereas a forward model of a dynamical system is a model of the transformation from inputs to outputs, an inverse model is a model of the transformation from outputs to inputs. Because the latter transformation need not be unique, there may be an infinite number of possible inverse models corresponding to any given dynamical system.[4] The forward model is generally unique.

Consider a dynamical system in the form of Equation 3:

$$\mathbf{y}[n+1] = h(\mathbf{x}[n], \mathbf{u}[n]). \tag{10}$$

Assuming that the transformation from $\mathbf{u}$ to $\mathbf{y}$ is a causal or many-to-one transformation, any system that produces $\mathbf{y}$ as a function h of $\mathbf{x}$ and $\mathbf{u}$ constitutes a forward model of the dynamical system. Thus a forward model

[4]The issue of *existence* of an inverse model will not play a significant role in this chapter; we will generally assume that an inverse exists. We also ignore the issue of how noise affects questions of existence and uniqueness.

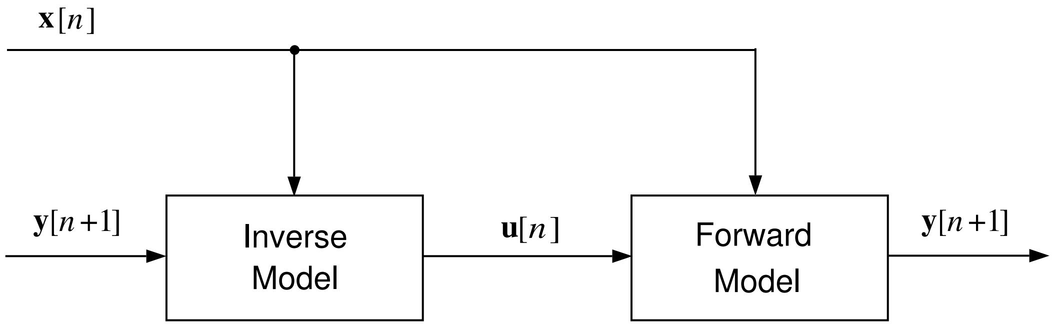

Figure 2: The mathematical relationship between forward models and inverse models.

is a mapping from inputs to outputs, in the context of a given state vector. Similarly, an inverse model is a mapping from outputs to inputs, again in the context of a given state vector. Mathematically, this relationship is expressed as follows:

$$\mathbf{u}[n] = h^{-1}(\mathbf{x}[n], \mathbf{y}[n+1]). \tag{11}$$

Note that we use the symbol h^{-1} even though this equation is not strictly speaking a mathematical inverse (it is not simply a swapping of the left and right sides of Equation 10). Nonetheless, Equation 11 is to be thought of as inverting the relationship between inputs and outputs in Equation 10, with the state thought of as a context. These relationships are summarized in Figure 2.

The terms "forward model" and "inverse model" are generally used to refer to internal models of dynamical systems and it is in this sense that we use the terms in the remainder of the chapter. Note that an internal model is a model of some particular dynamical system; thus, it is sensible to speak of an "approximate forward model" or an "approximate inverse model." It is also important to distinguish between actual values of variables and internal estimates of variables, and to distinguish between actual values of variables and desired values of variables. For example, a ball flying through the air has an actual position and velocity, but an internal model of the ball dynamics must work with internal estimates of position and velocity. Similarly, we must distinguish between the desired position of the ball and its actual or estimated position. We postpone a further discussion of these issues until later sections in which we see how forward models and inverse models can be used as components of control systems.

Control

The problem of controlling a dynamical system is essentially the problem of computing an input to the system that will achieve some desired behavior at its output. As we suggested earlier, computing an input from a desired output would intuitively seem to involve the notion of the inverse of the controlled system. In the next three sections we discuss the two principal kinds of control systems—feedforward control systems and feedback control systems. We will see that indeed both can be viewed in terms of the computation of an inverse.

Predictive control

Suppose that the system to be controlled—the *plant*—is currently believed to be in state $\hat{\mathbf{x}}[n]$. Suppose further that the desired output at the next time step is a particular vector $\mathbf{y}^*[n+1]$.[5] We wish to compute the control signal $\mathbf{u}[n]$ that will cause the plant to output a vector $\mathbf{y}[n+1]$ that is as close as possible to the desired vector $\mathbf{y}^*[n+1]$.[6] Clearly the appropriate computation to perform is that given by Equation 11; that is, we require an inverse model of the plant. An inverse model of the plant allows the control system to compute a control signal that is predicted to yield the desired future output. The use of an explicit inverse model of the plant as a controller is referred to as "predictive control." Predictive control comes in different varieties depending on the way the states are estimated.

Example

Let us consider the simple first-order plant given by the following next-state equation:

$$x[n+1] = .5x[n] + .4u[n] \tag{12}$$

and the following output equation:

$$y[n] = x[n]. \tag{13}$$

Substituting the next-state equation into the output equation yields the forward dynamic equation:

$$y[n+1] = .5x[n] + .4u[n]. \tag{14}$$

If the input sequence $u[n]$ is held at zero, then this dynamical system decays exponentially to zero, as shown in Figure 3(a).

[5] We will use the "hat" notation ($\hat{\mathbf{x}}$) throughout the paper to refer to *estimated* values of signals and the "asterisk" notation ($\mathbf{y}^*$) to refer to *desired* values of signals.

[6] This idea will be generalized in the section on model-reference control.

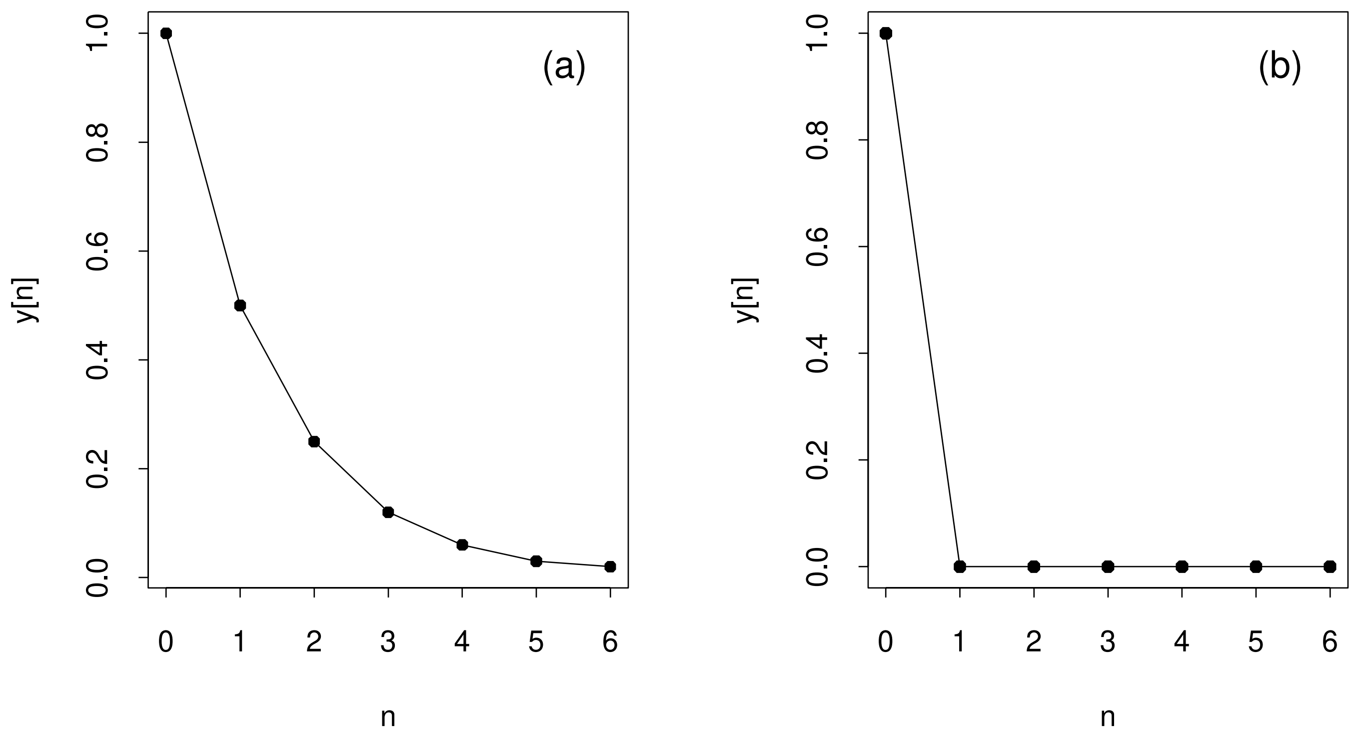

Figure 3: (a) The output of the uncontrolled dynamical system in Equation 14 with initial condition $x[0] = 1$. (b) The output of the dynamical system using the deadbeat controller in Equation 15. The desired output $y^*[n]$ is fixed at zero. The controller brings the actual output $y[n]$ to zero in a single time step.

A predictive controller for this system can be obtained by solving for $u[n]$ in Equation 14:

$$u[n] = -1.25\hat{x}[n] + 2.5y^*[n+1]. \tag{15}$$

Because this is an equation for a controller we treat y as the desired plant output rather than the actual plant output. Thus, the signal $y^*[n+1]$ is the input to the controller and denotes the desired future plant output. The controller output is $u[n]$. Note that we also assume that the state $x[n]$ must be estimated.

Suppose that the controller has a good estimate of the state and suppose that it is desired to drive the output of the dynamical system to a value d as quickly as possible. Setting $y^*[n+1]$ to d, letting $\hat{x}[n]$ equal $x[n]$, and substituting Equation 15 into Equation 14, we see that indeed the output y at time $n+1$ is equal to d (see Figure 3(b)). A predictive controller that can

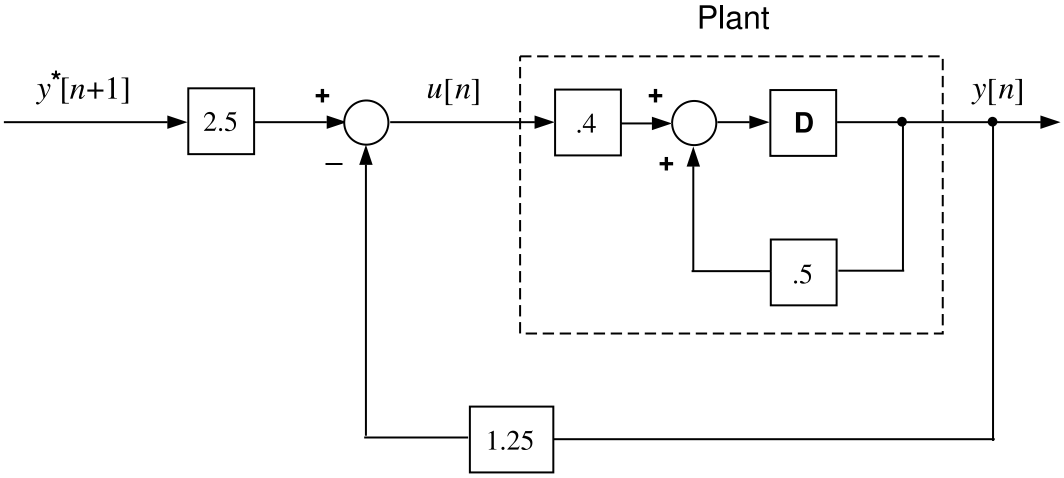

Figure 4: A deadbeat controller for the first-order example.

drive the output of a dynamical system to an arbitrary desired value in k time steps, where k is the order of the dynamical system (i.e., the number of state variables), is referred to as a *deadbeat* controller. In the following section, we provide a further example of a deadbeat controller for a second-order system.

How should the state be estimated in the deadbeat controller? Because the output equation (Equation 13) shows that the state and the output are the same, a natural choice for the state estimate is the current output of the plant. Thus, the controller can be written in the more explicit form:

$$u[n] = -1.25y[n] + 2.5y^*[n+1]. \tag{16}$$

This choice of state estimator is not necessarily the best choice in the more realistic case in which there are disturbances acting on the system, however. In the section on observers, we shall discuss a more sophisticated approach to state estimation. Even in this more general framework, however, the state estimate is generally computed based on feedback from the output of the plant. Thus a deadbeat controller is generally a feedback controller. An alternative approach to predictive control design—open-loop feedforward control—is discussed below.

Figure 4 shows the deadbeat controller for the first-order example. The symbol "D" in the figure refers to a one-time-step delay: A signal entering the delay is buffered for one time step. That is, if the signal on the right-hand side of the delay is $y[n]$, then the signal on the left-hand side of the delay is $y[n+1]$. It can be verified from the figure that $y[n+1]$ is equal to the sum of $.5y[n]$ and

$.4u[n]$, as required by the dynamic equations for this system (Equations 14 and 13).

A second-order example

Higher-order dynamical systems are characterized by the property that the control signal does not normally have an immediate influence on the output of the system, but rather exerts its influence after a certain number of time steps, the number depending on the order of the system. In this section we design a deadbeat controller for a second-order plant to indicate how this issue can be addressed.

Consider a second-order dynamical system with the following next-state equation:

$$\begin{pmatrix} x_1[n+1] \\ x_2[n+1] \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1[n] \\ x_2[n] \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u[n], \tag{17}$$

and output equation:

$$y[n] = \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} x_1[n] \\ x_2[n] \end{pmatrix}. \tag{18}$$

From the second row of Equation 17 it is clear that the control signal cannot affect the second state variable in one time step. This implies, from Equation 18, that the control signal cannot affect the output in a single time step. The control signal can affect the output in *two* time steps, however. Therefore we attempt to obtain a predictive controller in which the control signal is a function of the desired output two time steps in the future. Extracting the first component of the next-state equation and solving for $u[n]$ in terms of $x_2[n+1]$, we obtain the following:

$$u[n] = -a_1\hat{x}_1[n] - a_2\hat{x}_2[n] + y^*[n+2], \tag{19}$$

where we have used the fact that $x_1[n+1]$ is equal to $x_2[n+2]$ (from the second next-state equation) and $x_2[n+2]$ is equal to $y[n+2]$ (from the output equation). Although this equation relates the control signal at time n to the desired output at time $n+2$, there is a difficulty in estimating the states. In particular, $x_1[n]$ is equal to $y[n+1]$, thus we would need access to a future value of the plant output in order to implement this controller. As it stands, the controller is unrealizable.

To remove the dependence on the future value of the output in Equation 19, let us substitute the next-state equation into itself, thereby replacing $x_1[n]$ and

$x_2[n]$ with $x_1[n-1]$, $x_2[n-1]$ and $u[n-1]$. This substitution yields:

$$\begin{pmatrix} x_1[n+1] \\ x_2[n+1] \end{pmatrix} = \begin{pmatrix} {a_1}^2 + a_2 & a_1 a_2 \\ a_1 & a_2 \end{pmatrix} \begin{pmatrix} x_1[n-1] \\ x_2[n-1] \end{pmatrix} + \begin{pmatrix} a_1 \\ 1 \end{pmatrix} u[n-1] + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u[n].$$

Extracting the first component from this equation and solving for $u[n]$ yields:

$$u[n] = -({a_1}^2 + a_2)\hat{x}_1[n-1] - a_1 a_2 \hat{x}_2[n-1] - a_1 u[n-1] + y^*[n+2].$$

This equation depends only on quantities defined at time n or earlier. In particular, $\hat{x}_1[n-1]$ can be estimated by $y[n]$ and $\hat{x}_2[n-1]$ can be estimated by $y[n-1]$. This yields the following deadbeat controller:

$$u[n] = -({a_1}^2 + a_2)y[n] - a_1 a_2 y[n-1] - a_1 u[n-1] + y^*[n+2]. \tag{20}$$

The technique that we have described in this section is applicable to dynamical systems of any order. Because a state variable can always be expressed in terms of inputs and states at earlier moments in time, an unrealizable controller can always be converted into a realizable controller by expanding the next-state equation. It is also worth pointing out that the technique is also applicable to nonlinear systems, assuming that we are able to invert the equation relating the control signal to the future desired output signal. In cases in which this equation cannot be inverted analytically it may be possible to use numerical techniques.[7] These issues will arise again in the section on motor learning.

Open-loop feedforward control

The second class of predictive control systems is the class of open-loop feedforward control systems. Like the deadbeat controller, the open-loop feedforward controller is based on an explicit inverse model of the plant. The logic behind the open-loop feedforward controller is the same as behind the deadbeat controller: The controller computes a control signal which is predicted to yield a desired future output. The difference between the two approaches is the manner in which the state is estimated.

[7]It is also worth raising a cautionary flag: There is an important class of systems, including some linear systems, for which the techniques that we are discussing do not suffice and must be extended. Some systems are *uncontrollable*, which means that there are state variables that cannot be affected through a particular control variable. A proper treatment of this topic requires the notion of a *controllability gramian*. For further discussion, see Åström & Wittenmark (1984).

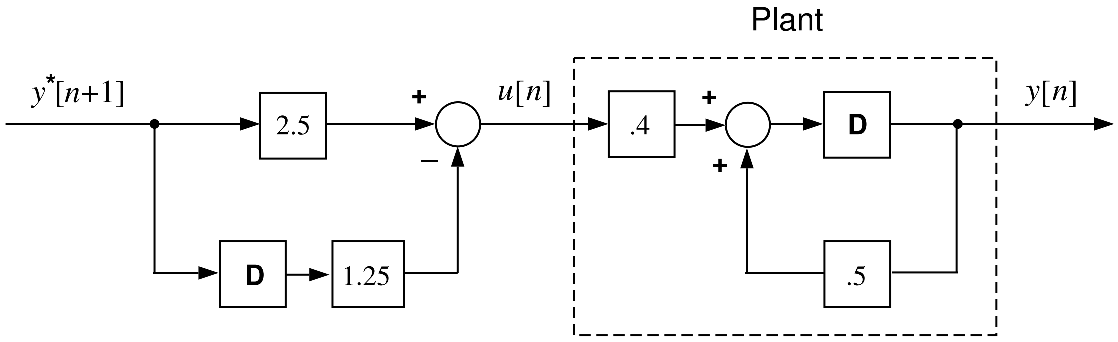

Figure 5: An open-loop feedforward controller for the first-order example.

Example

In the previous section, we saw that a deadbeat controller for the first-order plant has the form:

$$u[n] = -1.25y[n] + 2.5y^*[n+1],$$

where the signal $y[n]$ is considered to be an estimate of the state of the plant. We might also consider a controller of the following form:

$$u[n] = -1.25y^*[n] + 2.5y^*[n+1], \tag{21}$$

in which the state is estimated by the desired plant output $y^*[n]$ rather than the actual plant output $y[n]$. Figure 5 shows a diagrammatic representation of this controller. Note that there is no feedback from the plant to the controller; the loop from plant to controller has been "opened." Because of the lack of a feedback term in the control equation, the open-loop approach allows the entire control signal to be "preprogrammed" if the desired output trajectory is known in advance. The justification for the open-loop approach is that a good controller will keep the actual output and the desired output close together, so that replacing $y[n]$ by $y^*[n]$ in estimating the state may not incur much error. This is a strong assumption, however, because there are many sources of inaccuracy that can degrade the performance of a feedforward controller. In particular, if there are disturbances acting on the plant, then the state of the plant will diverge from the internal estimate of the state. Of course, no controller can control a system perfectly in the presence of disturbances. A system that utilizes feedback, however, has its state estimate continually reset and is therefore less likely to diverge significantly from reality than an open-loop controller. Another source of error is that the controller itself may be an inaccurate inverse model of the plant. Feedback renders the control system less sensitive to such inaccuracies.

One disadvantage of controllers based on feedback is that feedback can introduce stability problems. For this reason, open-loop feedforward controllers have important roles to play in certain kinds of control problems. As we discuss in a later section, stability is particularly of concern in systems with delay in the feedback pathway, thus an open-loop controller may be a reasonable option in such cases. Open-loop controllers also have an important role to play in composite control systems, when they are combined with an error-correcting feedback controller (see below). The division of labor into open-loop control and error-correcting control can be a useful way of of organizing complex control tasks.

Biological examples of feedforward control

There are many examples of open-loop feedforward control systems in the motor control literature. A particularly clear example is the vestibulo-ocular reflex (VOR). The VOR couples the movement of the eyes to the motion of the head, thereby allowing an organism to keep its gaze fixed in space. This is achieved by causing the the motion of the eyes to be equal and opposite to the motion of the head. The VOR control system is typically modeled as a transformation from head velocity to eye velocity (Robinson, 1981). The head velocity signal, provided by the vestibular system, is fed to a control system that provides neural input to the eye muscles. In our notation, the head velocity signal is the controller input $-\mathbf{y}^*[n]$, the neural command to the muscles is the control signal $\mathbf{u}[n]$, and the eye velocity signal is the plant output $\mathbf{y}[n]$. Note that the plant output (the eye velocity) has no effect on the control input (the head motion), thus the VOR is an open-loop feedforward control system. This implies that the neural machinery must implement an open-loop inverse model of the oculomotor plant. It is generally agreed in the literature on the VOR that such an inverse model exists in the neural circuitry and there have been two principal proposals for the neural implementation of the inverse model. Robinson (1981) has proposed a model based on an open-loop feedforward controller of the form shown earlier in Figure 5. In this model, as in the figure, the inverse model is implemented by adding the signals on a pair of parallel channels: a feed-through pathway and a pathway incorporating a delay (which corresponds to an integrator in Robinson's continuous-time model). An alternative model, proposed by Galliana and Outerbridge (1984), implements the inverse model by placing a forward model of the plant in an internal feedback pathway. (A closely related technique is described below;

see Figure 8).

Another interesting example of feedforward control arises in the literature on speech production. Lindblom, Lubker and Gay (1979) studied an experimental task in which subjects produced vowel sounds while their jaw was held open by a bite block. Lindblom et al. observed that the vowels produced by the subjects had formant frequencies in the normal range, despite the fact that unusual articulatory postures were required to produce these sounds. Moreover, the formant frequencies were in the normal range during the first pitch period, before any possible influence of acoustic feedback. This implies feedforward control of articulatory posture (with respect to the acoustic goal). Lindblom et al. proposed a qualitative model of this feedforward control system that again involved placing a forward model of the plant in an internal feedback pathway (see Figure 8).

Error-correcting feedback control

In this section we provide a brief overview of error-correcting feedback control systems. Error-correcting feedback control differs from predictive control in that it does not rely on an explicit inverse model of the plant. As we shall see, however, an error-correcting feedback control system can be thought of as implicitly computing an approximate plant inverse; thus, these two forms of control are not as distinct as they may seem.

An error-correcting feedback controller works directly to correct the error at the current time step between the desired output and the actual output. Consider the first order system presented earlier. A natural choice for an error-correcting feedback signal would be the weighted error:

$$u[n] = K(y^*[n] - y[n]), \tag{22}$$

where the scalar K is referred to as a *gain*. Note that the reference signal for this controller is the current desired output ($y^*[n]$) rather than the future desired output ($y^*[n+1]$) as in the predictive control approach. The performance of this feedback controller is shown in Figure 6 for several values of K. As K increases, the feedback controller brings the output of the plant to the desired value more rapidly.

A block diagram of the error-correctin feedback control system is shown in Figure 7, to be compared to the predictive controllers in Figure 4 and Figure 5. Several general distinctions can be drawn from comparing these control systems. One important distinction between predictive and error-correcting control is based on the temporal relationships that are involved. In predictive

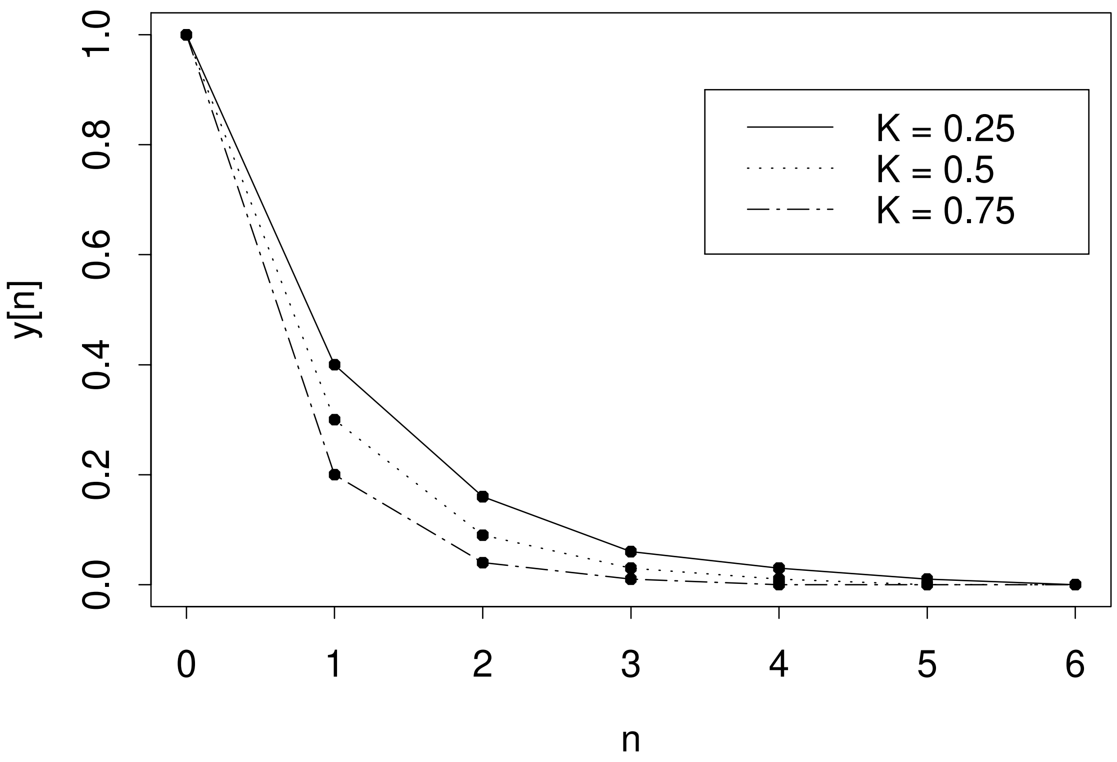

Figure 6: Performance of the error-correcting feedback controller as a function of the gain. The desired output $y^*[n]$ is fixed at zero.

control, the control signal is a function of the future desired output. If the predictive controller is a perfect inverse model of the plant, and if there are no unmodeled disturbances acting on the plant, then the future desired output will indeed be achieved by using the computed control signal. That is, an ideal predictive controller operates without error. An error-correcting feedback controller, on the other hand, corrects the error after the error has occurred, thus even under ideal conditions such a controller exhibits a certain amount of error. The assumption underlying error-correcting control is that the desired output changes relatively slowly; thus, correcting the error at the current time step is likely to diminish the error at the following time step as well. Another distinction between predictive control and error-correcting control has to do with the role of explicit knowledge about the plant. Predictive control requires explicit knowledge of the dynamics of the plant (a predictive controller is an inverse model of the plant). For example, the coefficients 1.25 and 2.5 in the predictive controllers in the previous section are obtained explicitly from knowledge of

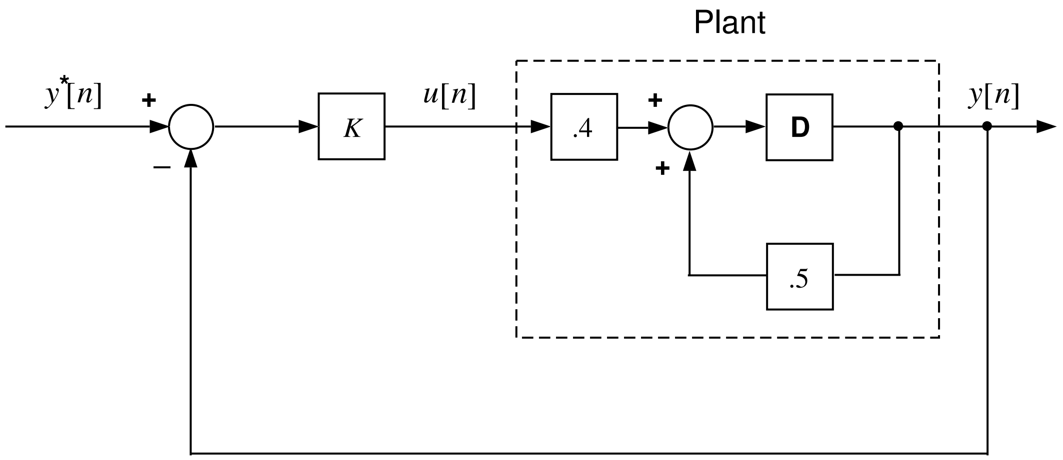

Figure 7: An error-correcting feedback controller for the first-order example.

the coefficients of the plant dynamic equation. Error-correcting control does not require the implementation of an explicit plant model. The design of an error-correcting controller (i.e., the choice of the feedback gain) generally depends on knowledge of the plant. However the knowledge that is required for such control design is often rather qualitative. Moreover, the performance of an error-correcting controller is generally rather insensitive to the exact value of the gain that is chosen. The predictive controllers based on feedback (i.e., the deadbeat controllers) are also somewhat insensitive to the exact values of their coefficients. This is in contrast to open-loop controllers, for which the performance is generally highly sensitive to the values of the coefficients. For example, choosing a value other than 2.5 in the forward path of the open-loop controller in the previous section yields a steady-state error at the output of the plant. Finally, as we have stated earlier, feedback controllers tend to be more robust to unanticipated disturbances than open-loop controllers.

Feedback control and plant inversion

Let us now establish a relationship between error-correcting feedback control and the notion of inverting a dynamical system. To simplify the argument we restrict ourselves to the first-order plant considered previously (Figure 7). Consider now the system shown in Figure 8, in which a replica of the plant is placed in a feedback path from the control signal to the error signal. This system is entirely equivalent to the preceding system, if we assume that there

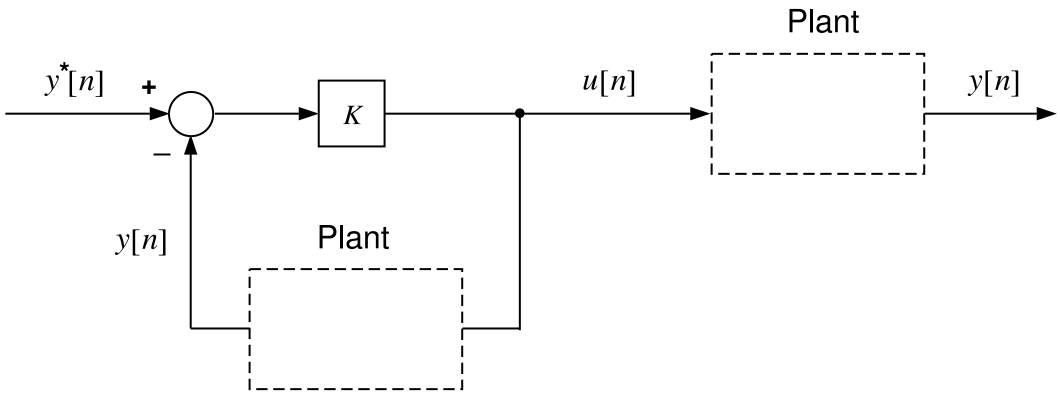

Figure 8: A control system in which the control signal is fed back through a replica of the plant. This system is mathematically equivalent to the feedback control system shown in Figure 7.

are no disturbances acting at the output of the plant. That is, the control signal in both diagrams is exactly the same:

$$u[n] = K(y^*[n] - y[n]).$$

This error equation can be expanded using the next-state equation (Equation 12) and the output equation (Equation 13):

$$u[n] = Ky^*[n] - .5Ky[n-1] - .4Ku[n-1].$$

Dividing by K and moving $u[n-1]$ to the left-hand side yields:

$$\frac{1}{K}u[n] + .4u[n-1] = y^*[n] - .5y[n-1].$$

If we now let the gain K go to infinity, the first term drops away, and we are left with an expression for $u[n-1]$:

$$.4u[n-1] = y^*[n] - .5y[n-1].$$

Shifting the time index and rearranging yields:

$$u[n] = -1.25y[n] + 2.5y^*[n+1].$$

This expression is an inverse dynamic model of the plant (cf. Equation 16).

What we have shown is that for large values of the gain, the internal loop in Figure 8 computes approximately the same control signal as an explicit inverse model of the plant.[8] Thus an error-correcting feedback control system with high gain is equivalent to an open-loop feedforward system that utilizes an explicit inverse model. This is true even though the feedback control system is clearly not computing an explicit plant inverse. We can think of the feedback loop as *implicitly* inverting the plant.

In many real feedback systems, it is impractical to allow the gain to grow large. One important factor that limits the magnitude of the gain is the presence of delays in the feedback loop, as we will see in the following section. Other factors have to do with robustness to noise and disturbances. It is also the case that some plants—so-called "nonminimum phase" plants—are unstable if the feedback gain is too large (Åström & Wittenmark, 1984). Nonetheless, it is still useful to treat a feedback control system with a finite gain K as computing an *approximation* to a inverse model of the plant. This approximation is ideally as close as possible to a true inverse of the plant, subject to constraints related to stability and robustness.

The notion that a high-gain feedback control system computes an approximate inverse of the plant makes intuitive sense as well. Intuitively, a high-gain controller corrects errors as rapidly as possible. Indeed, as we saw in Figure 6, as the gain of the feedback controller grows, its performance approaches that of a deadbeat controller (Figure 3(b)).

It is also worth noting that the system shown in Figure 8 can be considered in its own right as an implementation of a *feedforward* control system. Suppose that the replica of the plant in the feedback loop in Figure 8 is implemented literally as an internal forward model of the plant. If the forward model is an accurate model of the plant, then in the limit of high gain this internal loop is equivalent to an explicit inverse model of the plant. Thus the internal loop is an alternative implementation of a feedforward controller. The controller is an open-loop feedforward controller because there is no feedback from the actual plant to the controller. Note that this alternative implementation of an open-loop feedforward controller is consistent with our earlier characterization of feedforward control: (1) The control system shown in Figure 8 requires explicit knowledge of the plant dynamics (the internal forward model); (2) the performance of the controller is sensitive to inaccuracies in the plant model

[8] In fact, the mathematical argument just presented is not entirely correct. The limiting process in our argument is well-defined only if the discrete-time dynamical system is obtained by approximating an underlying continuous-time system and the time step of the approximation is taken to zero as the gain is taken to infinity. Readers familiar with the Laplace transform will be able to justify the argument in the continuous-time domain.

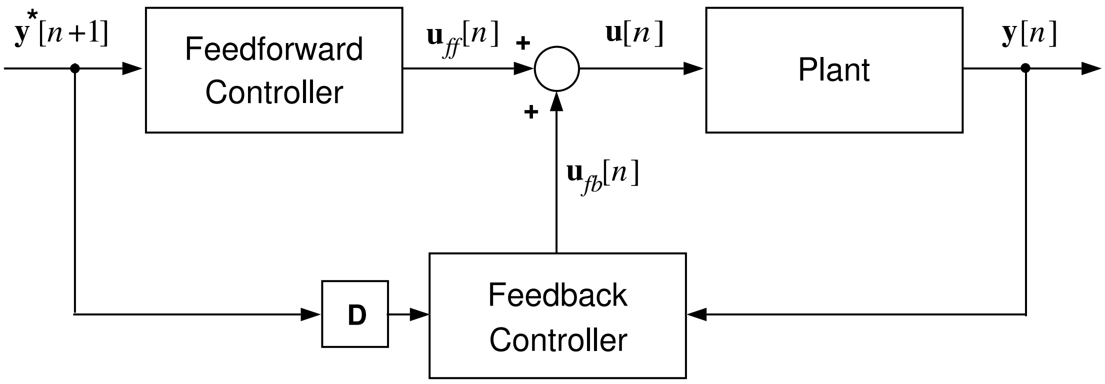

Figure 9: A composite control system composed of a feedback controller and a feedforward controller.

(the loop inverts the forward model, not the plant); and (3) the controller does not correct for unanticipated disturbances (there is no feedback from the actual plant output).

Composite control systems

Because feedforward control and error-correcting feedback control have complementary strengths and weaknesses, it is sensible to consider composite control systems that combine these two kinds of control. There are many ways that feedforward and feedback can be combined, but the simplest scheme—that of adding the two control signals—is generally a reasonable approach. Justification for adding the control signals comes from noting that because both kinds of control can be thought of as techniques for computing a plant inverse, the sum of the control signals is a sensible quantity.

Figure 9 shows a control system that is composed of a feedforward controller and an error-correcting feedback controller in parallel. The control signal in this composite system is simply the sum of the feedforward control signal and the feedback control signal:

$$\mathbf{u}[n] = \mathbf{u}_{ff}[n] + \mathbf{u}_{fb}[n].$$

If the feedforward controller is an accurate inverse model of the plant, and if there are no disturbances, then there is no error between the plant output and the desired output. In this case the feedback controller is automatically silent. Errors at the plant output, whether due to unanticipated disturbances or due to inaccuracies in the feedforward controller, are corrected by the feedback controller.

Delay

The delays in the motor control system are significant. Estimates of the delay in the visuomotor feedback loop have ranged from 100 to 200 milliseconds (Keele & Posner, 1968; Carlton, 1981). Such a large value of delay is clearly significant in reaching movements, which generally last from 250 milliseconds to a second or two.

Many artificial control systems are implemented with electrical circuits and fast-acting sensors, such that the delays in the transmission of signals within the system are insignificant when compared to the time constants of the dynamical elements of the plant. In such cases the delays are often ignored in the design and analysis of the controller. There are cases, however, including the control of processes in a chemical plant and the control of the flight of a spaceship, in which the delays are significant. In this section we make use of some of the ideas developed in the study of such problems to describe the effects of delays and to present tools for dealing with them.

What is the effect of delay on a control system? The essential effect of delay is that it generally requires a system to be operated with a small gain in the feedback loop. In a system with delay, the sensory reading that is obtained by the controller reflects the state of the system at some previous time. The control signal corresponding to that sensory reading may no longer be appropriate for the current state of the plant. If the closed-loop system is oscillatory, for example, then the delayed control signal can be out of phase with the true error signal and may contribute to the error rather than correct it. Such an out-of-phase control signal can destabilize the plant.

To illustrate the effect of delay in the closed loop, consider the first-order plant considered previously:

$$y[n+1] = .5y[n] + .4u[n],$$

which decays geometrically to zero when the control signal u is zero, as shown earlier in Figure 3(a). Let us consider four closed-loop control laws with delays of zero, one, two, and three time steps respectively. With no delay, we set $u[n] = -Ky[n]$, and the closed loop becomes:

$$y[n+1] = .5y[n] - .4Ky[n],$$

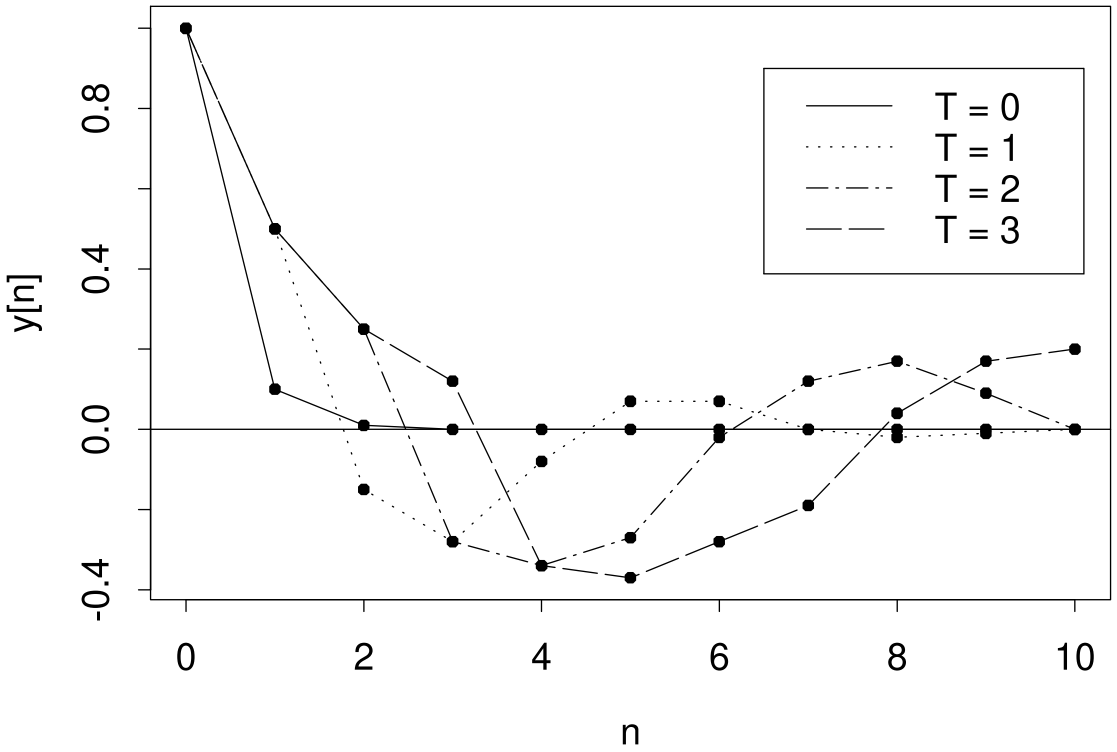

Figure 10: Performance of the feedback controller with delays in the feedback path.

where K is the feedback gain. Letting K equal 1 yields the curve labelled $T = 0$ in Figure 10(a), where we see that the regulatory properties of the system are improved when compared to the open loop. If the plant output is delayed by one time step, we obtain $u[n] = -Ky[n-1]$, and the closed-loop dynamics are given by

$$y[n+1] = .5y[n] - .4Ky[n-1],$$

which is a second-order difference equation. When K equals 1, the curve labelled $T = 1$ in Figure 10(b) is obtained, where we see that the delayed control signal has created an oscillation in the closed loop. As additional time steps of delay are introduced the closed loop becomes increasingly oscillatory and sluggish, as is shown by the curves labelled $T = 2$ and $T = 3$ in Figure 10(c). Eventually the closed-loop system becomes unstable. It is straightforward to solve for the maximum gain for which the closed loop remains stable at

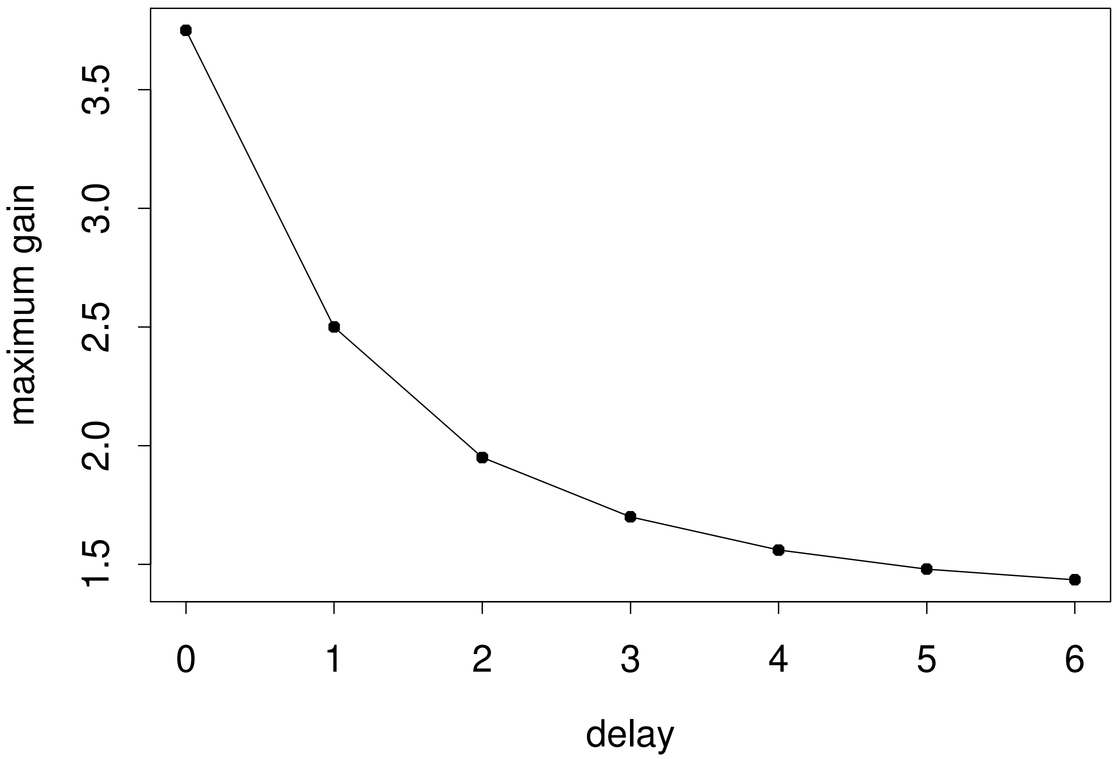

Figure 11: Maximum possible gain for closed-loop stability as a function of feedback delay.

each value of delay.[9] These values are plotted in Figure 11, where we see that the maximum permissible gain decreases as the delay increases. This plot illustrates a general point: To remain stable under conditions of delay, a closed-loop system must be operated at a lower feedback gain than a system without delay.

The Smith predictor

A general architecture for controlling a system with delay was developed by Smith (1958) and is referred to as a "Smith predictor." To understand the Smith predictor let us first consider some simpler approaches to dealing with delay. The simplest scheme is to simply utilize an open-loop feedforward con-

[9]These are the values of gain for which one of the roots of the characteristic equation of the closed-loop dynamics cross the unit circle in the complex plane (see, e.g., Åström & Wittenmark, 1984).

troller. If the feedforward controller is a reasonable approximation to an inverse of the plant then this scheme will control the plant successfully over short intervals of time. The inevitable disturbances and modeling errors will make performance degrade over longer time intervals, nonetheless, the advantages of feedforward control are not to be neglected in this case. By ignoring the output from the plant, the feedforward system is stable in spite of the delay. It might be hoped that a feedforward controller could provide coarse control of the plant without compromising stability, thereby bringing the magnitude of the performance errors down to a level that could be handled by a low-gain feedback controller.

Composite feedforward and feedback control is indeed the idea behind the Smith predictor, but another issue arises due to the presence of delay. Let us suppose that the feedforward controller is a perfect inverse model of the plant and that there are no disturbances. In this case there should be no performance errors to correct. Note, however, that the performance error cannot be based on the difference between the current reference signal and the current plant output, because the output of the plant is delayed by T time steps with respect to the reference signal. One approach to dealing with this problem would be to simply delay the reference signal by the appropriate amount before comparing it to the plant output. This approach has the disadvantage that the system is unable to anticipate potential future errors. Another approach—that used in the Smith predictor—is to utilize a forward model of the plant to predict the influence of the feedforward control signal on the plant, delay this prediction by the appropriate amount, and add it to the control signal to cancel the anticipated contribution of the feedback controller.

The control system now has both an inverse model for control and a forward model for prediction. Recall that one way to implement a feedforward controller is to utilize a forward model in an internal feedback loop (cf. Figure 8). Thus a forward model can be used both for implementing the feedforward controller and for predicting the plant output. Placing the forward model in the forward path of the control system yields the Smith predictor, as shown in Figure 12. Note that if the forward model and the delay model in the Smith predictor are perfect, then the outer loop in the diagram is cancelled by the positive feedback loop that passes through the forward model and the delay model. The remaining loop (the negative feedback loop that passes through the forward model) is exactly the feedforward control scheme described previously in connection with Figure 8. Because this internal loop is not subject to delay from the periphery, the feedback gain K can be relatively large and the inner loop can therefore provide a reasonable approximation to an inverse plant model.

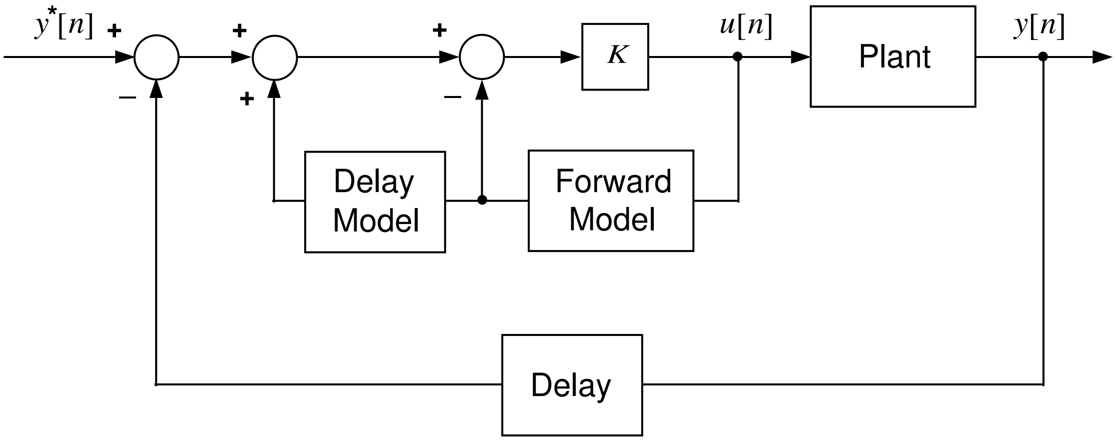

Figure 12: The Smith predictor.

Although the intuitions behind the Smith predictor have been present in the motor control literature for many years, only recently has explicit use been made of this technique in motor control modeling. Miall, Weir, Wolpert and Stein (in press) have studied visuomotor tracking under conditions of varying delay and have proposed a physiologically-based model of tracking in which the cerebellum acts as an adaptive Smith predictor.

Observers

In this section we briefly discuss the topic of state estimation. State estimation is a deep topic with rich connections to dynamical systems theory and statistical theory (Andersen & Moore, 1979). Our goal here is to simply provide some basic intuition for the problem, focusing on the important role of internal forward models in state estimation.

In the first-order example that we have discussed, the problem of state estimation is trivial because the output of the system is the same as the state (Equation 13). In most situations the output is a more complex function of the state. In such situations it might be thought that the state could be recovered by simply inverting the output function. There are two fundamental reasons, however, why this is not a general solution to the problem of state estimation. First, there are usually more state variables than output variables in dynamical models, thus the function g is generally not uniquely invertible. An example is the one-joint robot arm considered earlier. The dynamical model of the arm has two state variables: the joint angle at the current time step and the joint angle at the previous time step. There is a single output variable: the current joint angle. Thus the output function preserves information about only one of the state variables, and it is impossible to recover the state by simply inverting the output function. Minimally, the system must combine the outputs over two successive time steps in order to recover the state. The second reason why simply inverting the output function does not suffice is that in most situations there is stochastic uncertainty about the dynamics of the system as seen through its output. Such uncertainty may arise because of noise in the measurement device or because the dynamical system itself is a stochastic process. In either case, the only way to decrease the effects of the uncertainty is to average across several nearby time steps.

The general conclusion that arises from these observations is that robust estimation of the state of a system requires observing the output of the system over an extended period of time. State estimation is fundamentally a dynamic process.

To provide some insight into the dynamical approach to state estimation, let us introduce the notion of an *observer*. As shown in Figure 13, an observer is a dynamical system that produces an estimate of the state of a system based on observations of the inputs and outputs of the system. The internal structure of an observer is intuitively very simple. The state of the observer is the variable $\hat{\mathbf{x}}[n]$, the estimate of the state of the plant. The observer has access to the input to the plant, so it is in a position to predict the next state of the plant from the current input and its estimate of the current state. To make such a prediction the observer must have an internal model of the next-state function of the plant (the function f in Equation 1). If the internal model is accurate and if there is no noise in the measurement process or the state transition process, then the observer will accurately predict the next state of the plant. The observer is essentially an internal simulation of the plant dynamics that runs in parallel with the actual plant. Of course, errors will eventually accumulate in the internal simulation, thus there must be a way to couple the observer to the actual plant. This is achieved by using the plant output. Because the observer has access to the plant output, it is able to compare the plant output to its internal prediction of what the output should be, given its internal estimate of the state. To make such a prediction requires the observer to have an internal model of the output function of the plant (the function g in Equation 2). Errors in the observer's estimate of the state will be reflected in errors between the predicted plant output and the observed plant output. These errors can be used to correct the state estimate and thereby couple the observer dynamics to the plant dynamics.

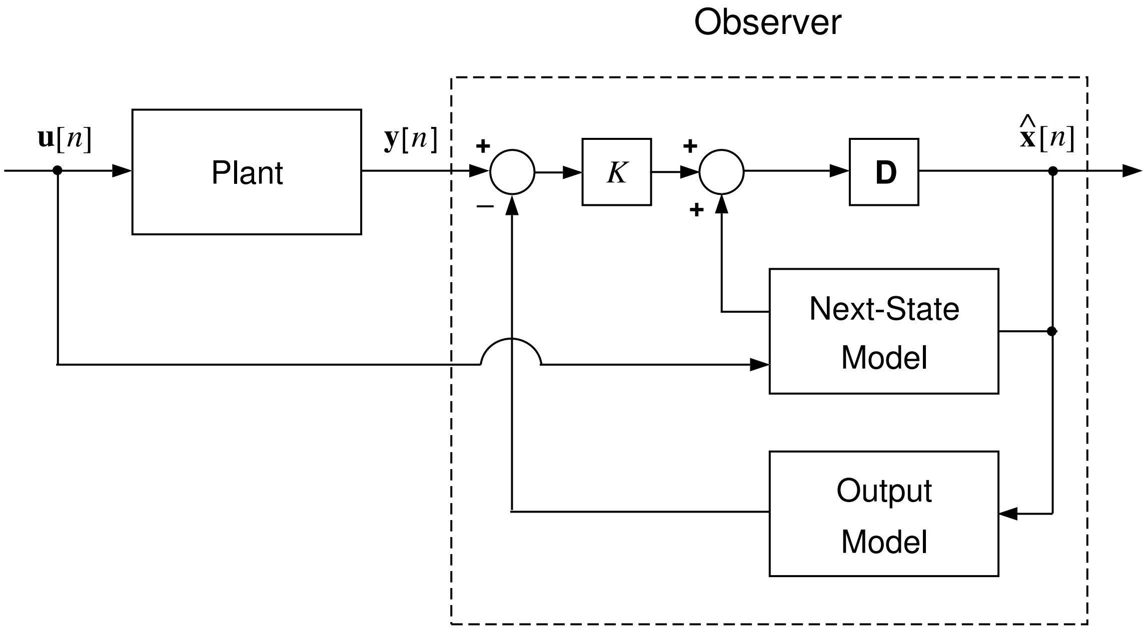

Figure 13: An observer. The inputs to the observer are the plant input and output and the output from the observer is the estimated state of the plant.

The internal state of the observer evolves according to the following dynamical equation:

$$\hat{\mathbf{x}}[n+1] = \hat{f}(\hat{\mathbf{x}}[n], \mathbf{u}[n]) + K(\mathbf{y}[n] - \hat{g}(\hat{\mathbf{x}}[n])), \tag{23}$$

where $\hat{f}$ and $\hat{g}$ are internal forward models of the next-state function and the output function, respectively. The first term in the equation is the internal prediction of the next state of the plant, based on the current state estimate and the known input to the plant. The second term is the coupling term. It involves an error between the actual plant output and the internal prediction of the plant output. This error is multiplied by a gain K, known as the *observer gain matrix*. The weighted error is then added to the first term to correct the state estimate.

In the case of linear dynamical systems, there is a well-developed theory to provide guidelines for setting the observer gain. In deterministic dynamical models, these guidelines provide conditions for maintaining the stability of the observer. Much stronger guidelines exist in the case of stochastic dynamical models, in which explicit assumptions are made about the probabilistic nature of the next-state function and the output function. In this case the observer is known as a "Kalman filter," and the observer gain is known as the "Kalman gain." The choice of the Kalman gain is based on the relative amount of noise in the next-state process and the output process. If there is relatively more noise in the output measurement process, then the observer should be conservative in changing its internal state on the basis of the output error and thus the gain K should be small. Conversely, if there is relatively more noise in the state transition process, then the gain K should be large. An observer with large K averages the outputs over a larger span of time, which makes sense if the state transition dynamics are noisy. The Kalman filter quantifies these tradeoffs and chooses the gain that provides an optimal tradeoff between the two different kinds of noise. For further information on Kalman filters, see Anderson and Moore (1979).

In the case of nonlinear dynamical systems, the theory of state estimation is much less well-developed. Progress has been made, however, and the topic of the nonlinear observer is an active area of research (Misawa & Hedrick, 1989).

Learning Algorithms

In earlier sections we have seen several ways in which internal models can be used in a control system. Inverse models are the basic building block of feedforward control. Forward models can also be used in feedforward control, and have additional roles in state estimation and motor learning. It is important to emphasize that an internal model is a form of knowledge about the plant. Many motor control problems involve interacting with objects in the external world, and these objects generally have unknown mechanical properties. There are also changes in the musculoskeletal system due to growth or injury. These considerations suggest an important role for adaptive processes. Through adaptation the motor control system is able to maintain and update its internal models of external dynamics.

The next several sections develop some of the machinery that can be used to understand adaptive systems. Before entering into the details, let us first establish some terminology and introduce a distinction. The adaptive algorithms that we will discuss are all instances of a general approach to learning known as *error-correcting learning* or *supervised learning.* A supervised learner is a system that learns a transformation from a set of inputs to a set of outputs. Examples of pairs of inputs and outputs are presented repeatedly to the learning system, and the system is required to abstract an underlying law or relation from this data so that it can generalize appropriately to new data.

Within the general class of supervised learning algorithms, there are two basic classes of algorithms that it is useful to distinguish: *regression* algorithms and *classification* algorithms. A regression problem involves finding a functional relationship between the inputs and outputs. The form of the relationship depends on the particular learning architecture, but generally it is real-valued and smooth. By way of contrast, a classification problem involves associating a category membership label with each of the input patterns. In a classification problem the outputs are generally members of a discrete set and the functional relationship from inputs to outputs is characterized by sharp decision boundaries.

The literature on supervised learning algorithms is closely related to the classical literature in statistics on regression and classification. Let us point out one salient difference between these traditions. Whereas statistical algorithms are generally based on processing a batch of data, learning algorithms are generally based on *on-line* processing. That is, a learning system generally cannot afford to wait for a batch of data to arrive, but must update its internal parameters immediately after each new learning trial.

The next two sections present two simple learning algorithms that are representative of classification algorithms and regression algorithms, respectively.

The perceptron

In this section we describe a simple classification learner known as the *perceptron* (Rosenblatt, 1962). The perceptron learns to assign a binary category label to each of a set of input patterns. For example, the input pattern might represent the output of a motion detection stage in the visual system and the binary label might specify whether or not an object can be caught before it falls to the ground. The perceptron is provided with examples of input patterns paired with their corresponding labels. The goal of the learning procedure is to extract information from the examples so that the system can generalize appropriately to novel data. That is, the perceptron must acquire a decision rule that allows it to make accurate classifications for those input patterns whose label is not known.

The perceptron is based on a thresholding procedure applied to a weighted sum. Let us represent the features of the input pattern by a set of real numbers $x_1, x_2, \ldots, x_n$. For each input value x_i there is a corresponding *weight* w_i. The perceptron sums up the weighted feature values and compares the weighted sum to a threshold θ. If the sum is greater than the threshold, the output is one, otherwise the output is zero. That is, the binary output y is computed

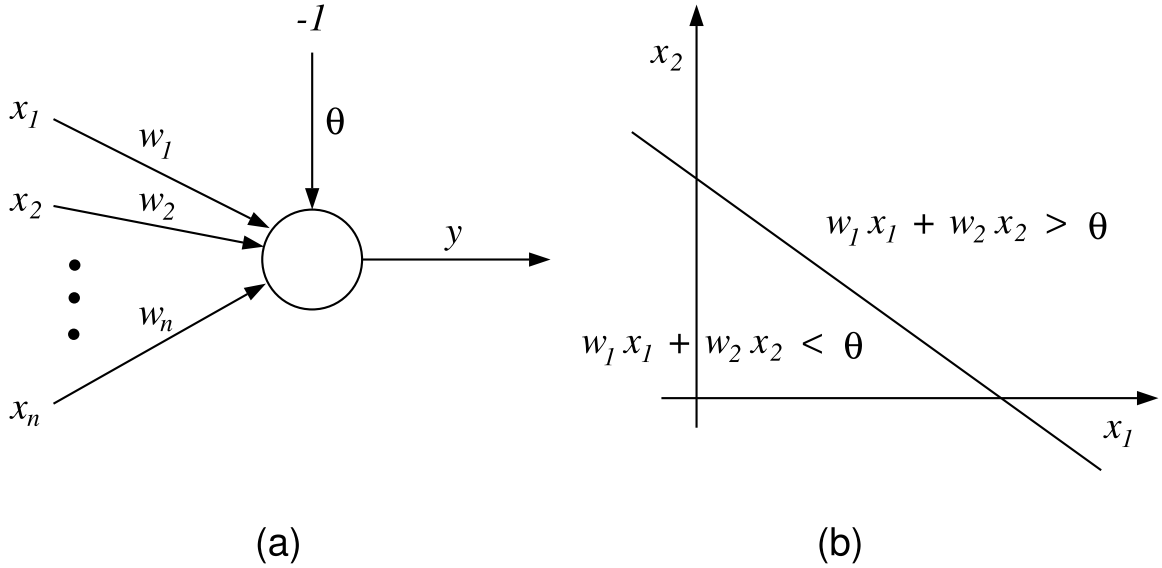

Figure 14: (a) A perceptron. The output y is obtained by thresholding the weighted sum of the inputs. The threshold θ can be treated as a weight emanating from an input line whose value is fixed at -1 (see below). (b) A geometric representation of the perceptron in the case of two input values x_1 and x_2. The line $w_1x_1 + w_2x_2 = \theta$ is the *decision surface* of the perceptron. For points lying above the decision surface the output of the perceptron is one. For points lying below the decision surface the output of the perceptron is zero. The parameters w_1 and w_2 determine the slope of the line and the parameter θ determines the offset of the decision surface from the origin.

as follows:

$$y = \begin{cases} 1 & \text{if } w_1x_1 + w_2x_2 + \cdots + w_nx_n > \theta \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

The perceptron can be represented diagrammatically as shown in Figure 14(a).

The perceptron learning algorithm is a procedure that changes the weights w_i as a function of the perceptron's performance on the training examples. To describe the algorithm, let us assume for simplicity that the input values x_i are either zero or one. We represent the binary category label as y^*, which also is either zero or one. There are four cases to consider. Consider first the case in which the desired output y^* is one, but the actual output y is zero. There are two ways in which the system can correct this error: either the threshold can be lowered or the weighted sum can be increased. To increase the weighted sum it suffices to increase the weights. Note, however, that it is of no use to increase the weights on the input lines that have a zero input value, because those lines do not contribute to the weighted sum. Indeed it is sensible to leave the weights unchanged on those lines so as to avoid disturbing the settings that have been made for other patterns. Consider now the case in which the desired output y^* is zero, but the actual output y is one. In this case the weighted sum is too large and needs to be decreased. This can be accomplished by increasing the threshold and/or decreasing the weights. Again the weights are changed only on the active input lines. The remaining two cases are the cases in which the desired output and the actual output are equal. In these cases, the perceptron quite reasonably makes no changes to the weights of the threshold.

The algorithm that we have described can be summarized in a single equation. The change to a weight w_i is given by:

$$\Delta w_i = \mu(y^* - y)x_i, \tag{25}$$

where μ is a small positive number referred to as the *learning rate*. Note that in accordance with the description given above, changes are made only to those weights that have a nonzero input value x_i. The change is of the appropriate sign due to the $(y^* - y)$ term. A similar rule can be written for the threshold θ:

$$\Delta\theta = -\mu(y^* - y), \tag{26}$$

which can be treated as a special case of the preceding rule if we treat the threshold as a weight emanating from an input line whose value is always -1.

Geometrically, the perceptron describes a hyperplane in the n-dimensional space of the input features, as shown in Figure 14(b). The perceptron learning algorithm adjusts the position and orientation of the hyperplane to attempt to place all of the input patterns with a label of zero on one side of the hyperplane and all of the input patterns with a label of one on the other side of the hyperplane. It can be proven that the perceptron is guaranteed to find a solution that splits the data in this way, if such a solution exists (Duda & Hart, 1973).

The LMS algorithm

The perceptron is a simple, on-line scheme for solving classification problems. What the perceptron is to classification, the Least Mean Squares (LMS) algorithm is to regression (Widrow & Hoff, 1960). In this section we derive the

LMS algorithm from the point of view of optimization theory. We shall see that it is closely related to the perceptron algorithm.

The LMS algorithm is essentially an on-line scheme for performing multivariate linear regression. Recall that the supervised learning paradigm involves the repeated presentation of pairs of inputs and desired outputs. In classification the desired outputs are binary, whereas in regression the desired outputs are real-valued. For simplicity let us consider the case in which a multivariate input vector is paired with a single real-valued output (we consider the generalization to multiple real-valued outputs below.) In this case, the regression surface is a $n+1$-dimensional hyperplane, where n is the number of input variables. The equation describing the hyperplane is as follows:

$$y = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b, \tag{27}$$

where the *bias* b allows the hyperplane to have a non-zero intercept along the y-axis. The bias is the analog of the negative of the threshold in the perceptron.

The regression equation (Equation 27) can be computed by the simple processing unit shown in Figure 15(a). As in the case of the perceptron, the problem is to develop an algorithm for adjusting the weights and the bias of this processing unit based on the repeated presentation of input-output pairs. As we will see, the appropriate algorithm for doing this is exactly the same as the algorithm developed for the perceptron (Equations 25 and 26). Rather than motivate the algorithm heuristically as we did in the previous section, let us derive the algorithm from a different perspective, introducing the powerful tools of optimization theory. We consider a *cost function* that measures the discrepancy between the actual output of the processing unit and the desired output. In the case of the LMS algorithm this cost function is one-half the squared difference between the actual output y and the desired output y^*:

$$J = \frac{1}{2}(y^* - y)^2. \tag{28}$$

Note that J is a function of the parameters w_i and b (because y is a function of these parameters). J can therefore be optimized (minimized) by proper choice of the parameters. We first compute the derivatives of J with respect to the parameters; that is, we compute the *gradient* of J with respect to w_i and b:

$$\begin{aligned} \frac{\partial J}{\partial w_i} &= -(y^* - y)\frac{\partial y}{\partial w_i} && (29) \\ &= -(y^* - y)x_i && (30) \end{aligned}$$

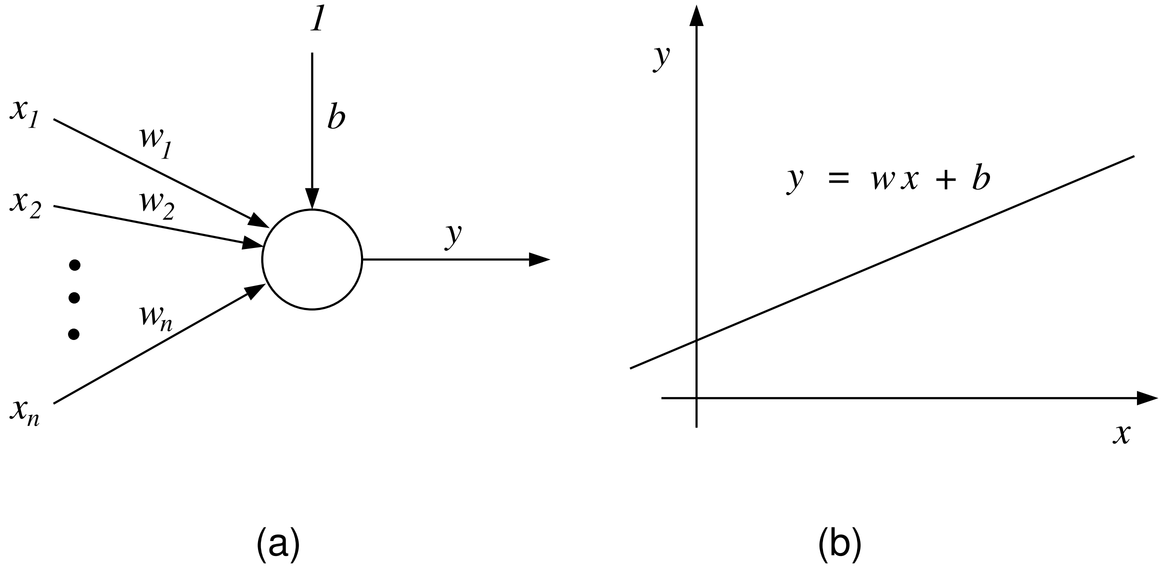

Figure 15: (a) An LMS processing unit. The output y is obtained as a weighted sum of the inputs. The bias can be treated as a weight emanating from an input line whose value is fixed at 1. (b) A geometric representation of the LMS unit in the case of a single input value x. The output function is $y = wx + b$, where the parameter w is the slope of the regression line and the parameter b is the y-intercept.

and

$$\begin{aligned} \frac{\partial J}{\partial b} &= -(y^* - y)\frac{\partial y}{\partial b} && (31) \\ &= -(y^* - y). && (32) \end{aligned}$$

The gradient points in the direction in which J increases most steeply (see Figure 16); therefore, to decrease J we take a step in the direction of the negative of the gradient:

$$\Delta w_i = \mu(y^* - y)x_i \tag{33}$$

and

$$\Delta b = \mu(y^* - y), \tag{34}$$

where μ is the size of the step. Note that we have recovered exactly the equations that were presented in the previous section (Equations 25 and 26). The difference between these sets of equations is the manner in which y is computed. In Equations 33 and 34, y is a linear function of the input variables (Equation 27), whereas in Equations 25 and 26, y is a binary function of

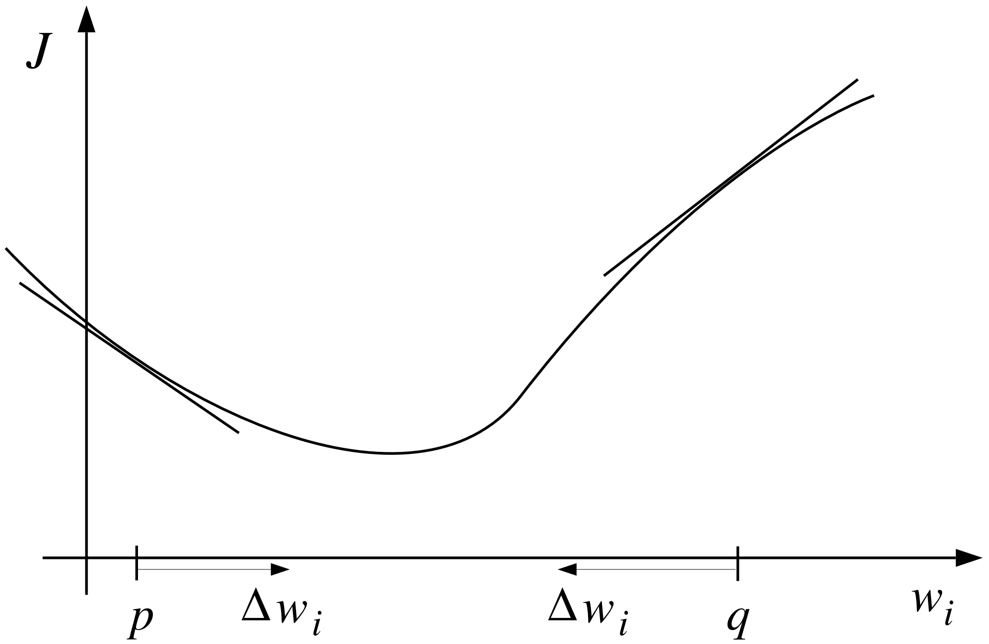

Figure 16: The logic of gradient descent: If the derivative of J with respect to w_i is positive (as it is at q), then to decrease J we decrease w_i. If the derivative of J with respect to w_i is negative (as it is at p), we increase w_i. The step Δw_i also depends on the magnitude of the derivative.

the input variables (Equation 24). This seemingly minor difference has major implications—the LMS algorithm (Equations 27, 33 and 34) and the perceptron algorithm (Equations 24, 25 and 26) have significantly different statistical properties and convergence properties, reflecting their differing roles as a regression algorithm and a classification algorithm, respectively. For an extensive discussion of these issues see Duda and Hart (1973).

Although we have presented the LMS algorithm and the perceptron learning algorithm in the case of a single output unit, both algorithms are readily extended to the case of multiple output units. Indeed, no new machinery is required—we simply observe that each output unit in an array of output units has its own set of weights and bias (or threshold), so that each output unit learns independently and in parallel. In the LMS case, this can be seen formally as follows. Let us define a multi-output cost function:

$$J = \frac{1}{2}\|\mathbf{y}^* - \mathbf{y}\|^2 = \frac{1}{2}\sum_i (y_i^* - y_i)^2, \tag{35}$$

where y_i^* and y_i are the i^{th} components of the desired output vector and the actual output vector, respectively. Letting w_{ij} denote the weight from input unit j to output unit i, we have:

$$\begin{aligned}\frac{\partial J}{\partial w_{ij}} &= \sum_k \frac{\partial J}{\partial y_k}\frac{\partial y_k}{\partial w_{ij}} &(36)\\ &= -(y_i^* - y_i)x_j, &(37)\end{aligned}$$

which shows that the derivative for weight w_{ij} depends only on the error at output unit i.

Nonlinear learning algorithms

The LMS algorithm captives in a simple manner many of the intuitions behind the notion of the *motor schema* as discussed by Schmidt (1975), Koh and Meyer (1991) and others. A motor schema is an internal model that utilizes a small set of parameters to describe a family of curves. The parameters are adjusted incrementally as a function of experience so that the parameterized curve approximates a sensorimotor transformation. The incremental nature of the approximation implies that the motor schema tends to generalize best in regions of the input space that are nearby to recent data points and generalize less well for regions that are further from recent data points. Moreover, the ability of the system to generalize can often be enhanced if the data points are somewhat spread out in the input space than if they are tightly clustered. All of these phenomena are readily observed in the performance of the LMS algorithm.

Although the LMS algorithm and the perceptron are serviceable for simple models of adaptation and learning, they are generally too limited for more realistic cases. The difficulty is that many sensorimotor systems are nonlinear systems and the LMS algorithm and the perceptron are limited to learning linear mappings. There are many ways to generalize the linear approach, however, to treat the problem of the incremental learning of nonlinear mappings. This is an active area of research in a large number of disciplines and the details are beyond the scope of this paper (see, e.g., Geman, Bienenstock, & Doursat, 1992). Nonetheless it is worth distinguishing a few of the trends. One general approach is to consider systems that are nonlinear in the inputs, but linear in the parameters. An example of such a system would be a polynomial:

$$y = ax^3 + bx^2 + cx + d, \tag{38}$$

where the coefficients a, b, c and d are the unknown parameters. By defining a new set of variables $z_1 = x^3$, $z_2 = x^2$, and $z_3 = x$, we observe that this system is linear in the parameters and also linear in the transformed set of variables. Thus an LMS processing unit can be used after a pre-processing level in which a fixed set of nonlinear transformations are applied to the input x. There are two difficulties with this approach—first, in cases with more than a single input variable, the number of cross-products (e.g., $x_1x_5x_8$) increase exponentially; and second, high-order polynomials tend to oscillate wildly between the data points, leading to poor generalization (Duda & Hart, 1973).

A second approach which also does not stray far from the linear framework is to use *piecewise* linear approximations to nonlinear functions. This approach generally requires all of the data to be stored so that the piecewise fits can be constructed on the fly (Atkeson, 1990). It is also possible to treat the problem of splitting the space as part of the learning problem (Jordan & Jacobs, 1992).

Another large class of algorithms are both nonlinear in the inputs and nonlinear in the parameters. These algorithms include the generalized splines (Wahba, 1990, Poggio & Girosi, 1990), the feedforward neural network (Hinton, 1989), and regression trees (Breiman, Friedman, Olshen, & Stone, 1984; Friedman, 1990; Jordan & Jacobs, 1992). For example, the standard two-layer feedforward neural network can be written in the form:

$$y_i = f(\sum_j w_{ij} f(\sum_k v_{jk} x_k)), \tag{39}$$

where the parameters w_{ij} and v_{jk} are the weights of the network and the function f is a fixed nonlinearity. Because the weights v_{jk} appear "inside" the nonlinearity, the system is nonlinear in the parameters and a generalization of the LMS algorithm, known as "backpropagation," is needed to adjust the parameters (Rumelhart, Hinton, & Williams, 1986; Werbos, 1974). The generalized splines and the regression trees do not utilize backpropagation, but rather make use of other forms of generalization of the LMS algorithm.

A final class of algorithms are the non-parametric approximators (e.g., Specht, 1991). These algorithms are essentially smoothed lookup tables. Although they do not utilize a parameterized family of curves, they nonetheless exhibit generalization and interference due to the smoothing.

In the remainder of this chapter, we lump all of these various nonlinear learning algorithms into the general class of supervised learning algorithms. That is, we simply assume the existence of a learning algorithm that can acquire a nonlinear mapping based on samples of pairs of inputs and corresponding outputs. The diagram that we use to indicate a generic supervised learning algorithm is shown in Figure 17. As can be seen, the generic supervised learning system has an input $\mathbf{x}$, an output $\mathbf{y}$, and a desired output $\mathbf{y}^*$. The error between the desired output and the actual output is used by

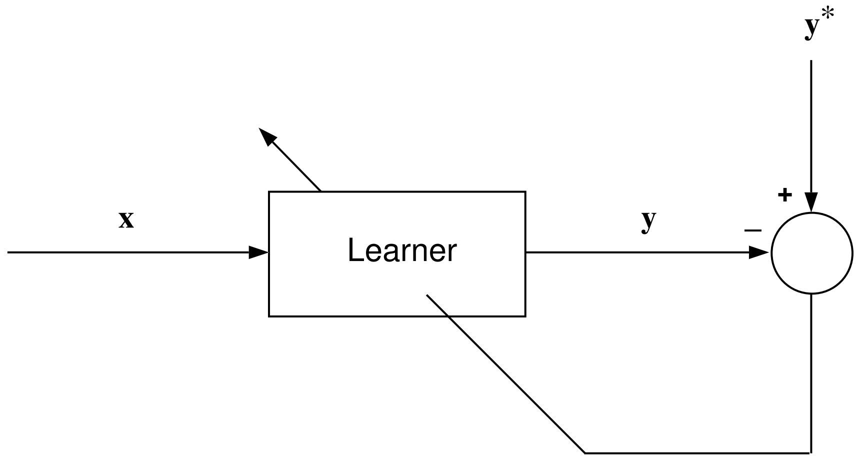

Figure 17: A generic supervised learning system.

the learning algorithm to adjust the internal parameters of the learner. This adjustment process is indicated by the diagonal arrow in the figure.

Motor Learning

In this section we put together several of the ideas that have been introduced in earlier sections and discuss the problem of motor learning. To fix ideas, we consider feedforward control; in particular, we discuss the problem of learning an inverse model of the plant (we discuss a more general learning problem in the following section). We distinguish between two broad approaches to learning an inverse model—a direct approach that we refer to as *direct inverse modeling*, and an indirect approach that we refer to as *distal supervised learning.* We also describe a technique known as *feedback error learning* that combines aspects of the direct and indirect approaches. All three approaches acquire an inverse model based on samples of inputs and outputs from the plant. Whereas the direct inverse modeling approach uses these samples to train the inverse model directly, the distal supervised learning approach trains the inverse model indirectly, through the intermediary of a learned forward model of the plant. The feedback error learning approach also trains the inverse model directly, but makes use of an associated feedback controller to provide an error signal.

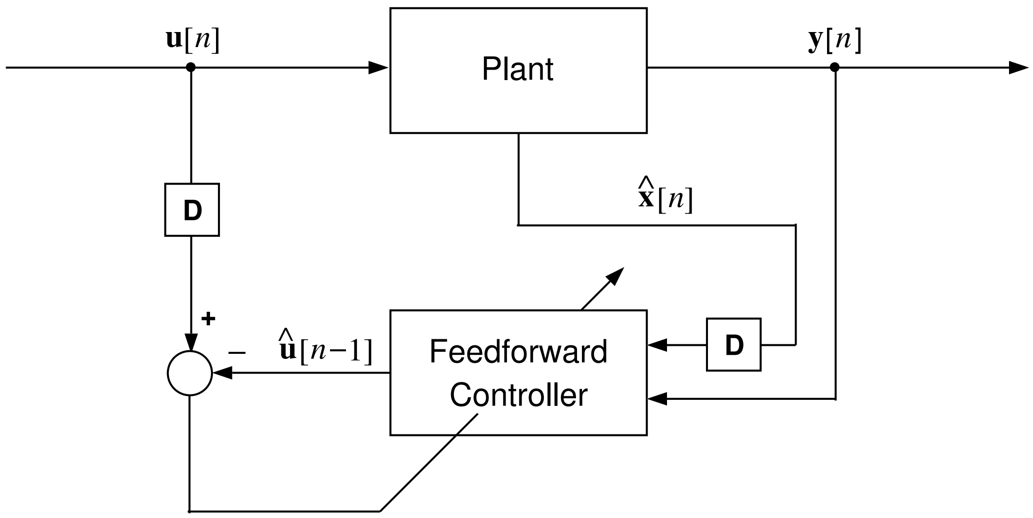

Figure 18: The direct inverse modeling approach to learning a feedforward controller. The state estimate $\hat{\mathbf{x}}[n]$ is assumed to be provided by an observer (not shown).

Direct inverse modeling

How might a system acquire an inverse model of the plant? One straightforward approach is to present various test inputs to the plant, observe the outputs, and provide these input-output pairs as training data to a supervised learning algorithm by reversing the role of the inputs and the outputs. That is, the plant output is provided as an input to the learning controller, and the controller is required to produce as output the corresponding plant input. This approach, shown diagrammatically in Figure 18, is known as direct inverse modeling (Widrow & Stearns, 1985, Atkeson & Reinkensmeyer, 1988; Kuperstein, 1988; Miller, 1987). Note that we treat the plant output as being observed at time n. Because an inverse model is a relationship between the state and the plant input at one moment in time with the plant output at the following moment in time (cf. Equation 11), the plant input ($\mathbf{u}[n]$) and the state estimate ($\hat{\mathbf{x}}[n]$) must be delayed by one time step to yield the proper temporal relationships. The input to the learning controller is therefore the current plant output $\mathbf{y}[n]$ and the delayed state estimate $\hat{\mathbf{x}}[n-1]$. The controller is required to produce the plant input that gave rise to the current output, in the context of the delayed estimated state. This is generally achieved by the optimization of the following sum-of-squared-error cost function:

$$J = \frac{1}{2}\|\mathbf{u}[n-1] - \hat{\mathbf{u}}[n-1]\|^2, \tag{40}$$

where $\hat{\mathbf{u}}[n-1]$ denotes the controller output.

Example

Consider the first-order plant:

$$y[n+1] = .5x[n] + .4u[n]. \tag{41}$$

As we have seen previously, the inverse model for this plant is linear in the estimated state and the desired output (cf. Equation 15). Let us assume that we do not know the appropriate values for the coefficients in the inverse model, thus we replace them with unknown values v_1 and v_2:

$$\hat{u}[n] = v_1\hat{x}[n] + v_2 y[n+1]. \tag{42}$$

This equation is linear in the unknown parameters, thus we can use the LMS algorithm to learn the values of v_1 and v_2. We first shift the time index in Equation 42 to write the inverse model in terms of the current plant output ($y[n]$). This requires delaying the control input and the state estimate by one time step. Connecting the LMS unit to the plant with the appropriate delays in place yields the wiring diagram in Figure 19. Note that we assume that the state is estimated by feedback from the plant, thus, the delayed state $x[n-1]$ is estimated by the delayed plant output $y[n-1]$ (cf. Equation 16). The inputs to the LMS processing unit are the plant output $y[n]$ and the delayed plant output $y[n-1]$. The target for the LMS unit is the delayed plant input $u[n-1]$. Note that if the unit is equipped with a bias, the bias value will converge to zero because it is not needed to represent the inverse model for this plant.

The nonconvexity problem

The direct inverse modeling approach is well-behaved for linear systems and indeed can be shown to converge to correct parameter estimates for such systems under certain conditions (Goodwin & Sin, 1984). For nonlinear systems, however, a difficulty arises that is related to the general "degrees-of-freedom problem" in motor control (Bernstein, 1967). The problem is due to a particular form of redundancy in nonlinear systems (Jordan, 1992). In such systems, the "optimal" parameter estimates (i.e., those that minimize the cost function in Equation 40) in fact yield an incorrect controller.

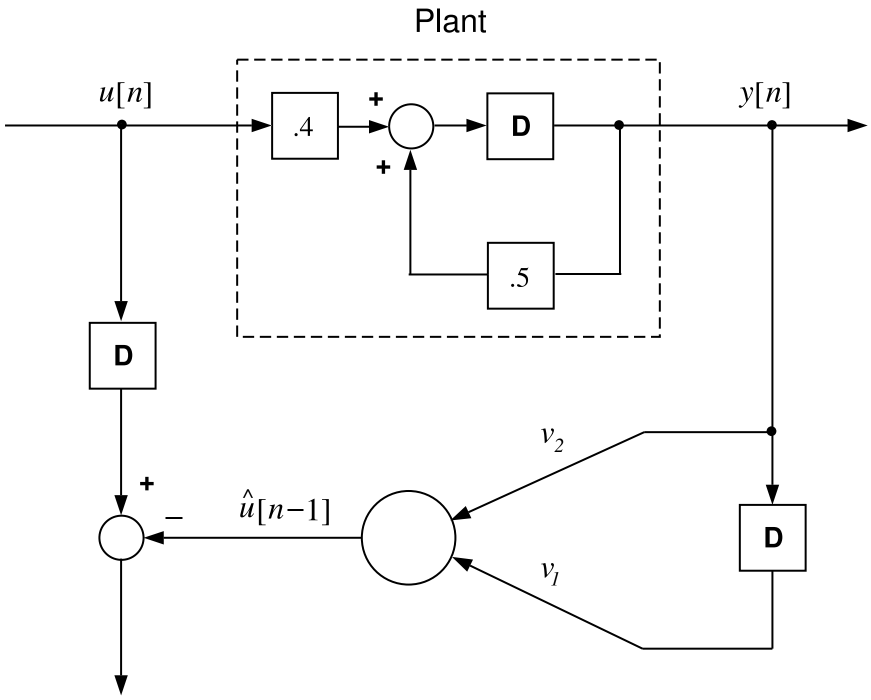

Figure 19: An example of the direct inverse modeling approach. An LMS processing unit is connected to the first order plant. The bias has been omitted for simplicity.

To illustrate, let us consider the planar kinematic arm shown in Figure 20. The arm has three joint angles, which we denote by θ_1, θ_2, and θ_3. The tip of the arm can be described by a pair of Cartesian coordinates, which we denote by y_1 and y_2. For every vector of joint angles $\boldsymbol{\theta}$ there is a corresponding Cartesian position vector $\mathbf{y}$. The mapping from $\boldsymbol{\theta}$ to $\mathbf{y}$ is a nonlinear function known as the *forward kinematics* of the arm.

Suppose that we use the direct inverse modeling approach to learn the *inverse kinematics* of the arm; that is, the mapping from $\mathbf{y}$ to $\boldsymbol{\theta}$ (cf. Kuperstein, 1988). Data for the learning algorithm are obtained by trying random joint angle configurations and observing the corresponding position of the tip of the arm. A nonlinear supervised learning algorithm is used to learn the mapping from tip positions to joint angles. Figure 21 shows the results of a simulation of this approach for the planar arm. The figure is an error vector field, that is, the tail of each arrow is a desired position, and the head of each arrow is the position produced by utilizing the inverse model to produce a set of

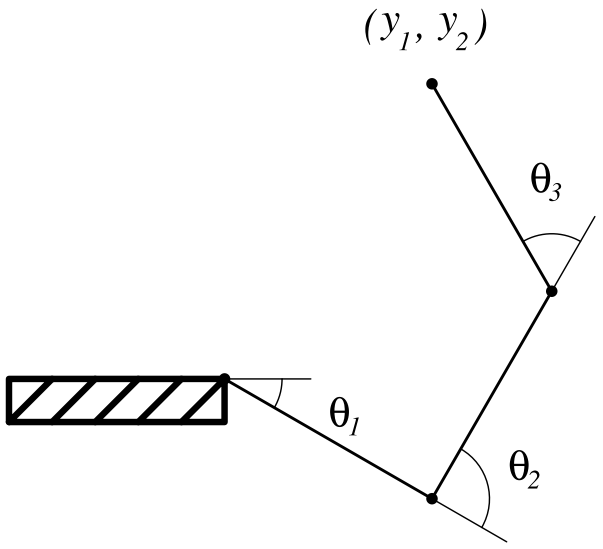

Figure 20: A three-joint planar arm.

joint angles. As can be observed, there are substantial errors throughout the workspace.

It is possible to rule out a number of possible explanations for the errors. The errors are not explained by possible local minima, by insufficient training time, or by poor approximation capability of the inverse model (Jordan & Rumelhart, 1992). The particular inverse model used in the simulation was a feedforward neural network trained with backpropagation, but it can be shown that any least-squares-based nonlinear approximator would give a similar result. To understand the difficulty, let us consider the direct inverse modeling approach geometrically, as shown in Figure 22. The figure shows the joint space on the left and the Cartesian space on the right. The arm is a redundant kinematic system; that is, to every tip position inside the workspace, there are an infinite set of joint angle configurations that achieve that position. Thus, to every point on the right side of the figure, there is a corresponding region (the *inverse image*) on the left. The direct inverse modeling approach samples randomly in joint space, observes the corresponding points in Cartesian space, and learns the mapping in the reverse direction. Let us suppose that three sample points happen to fall in a particular inverse image (see Figure 22). All three of these points correspond to a single point in Cartesian space, thus, the direct inverse learner is presented with data that are one-to-many: A single input maps to three different target outputs. The optimal

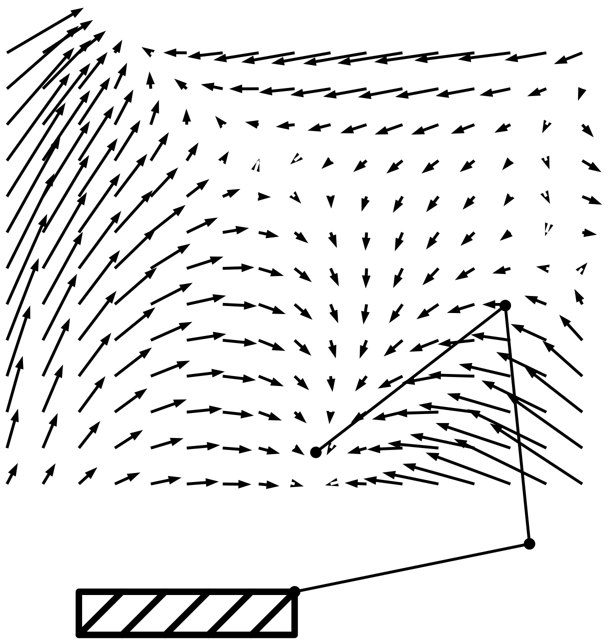

Figure 21: Near-asymptotic performance of direct inverse modeling. Each vector represents the error at a particular position in the workspace.

least-squares solution is to produce an output that is an average of the three targets. If the inverse image has a nonconvex shape, as shown in the figure, then the average of the three targets lies outside of the inverse image and is therefore not a solution.

It is easy to demonstrate that linear systems always have convex inverse images, thus the nonconvexity problem does not arise for such systems.[10] The problem does arise for nonlinear systems, however. In particular, Figure 23 demonstrates that the problem arises for the planar kinematic arm. The figure shows two particular joint angle configurations that lie in the same inverse

[10]Let $\mathbf{y} = f(\mathbf{x})$ be a linear function, and consider a particular point $\mathbf{y}^*$ in the range of f. The convex combination of any two points $\mathbf{x}_1$ and $\mathbf{x}_2$ that lie in the inverse image of $\mathbf{y}^*$ also lies in the inverse image of $\mathbf{y}^*$:

$$\begin{aligned} f(\alpha\mathbf{x}_1 + (1-\alpha)\mathbf{x}_2) &= \alpha f(\mathbf{x}_1) + (1-\alpha)f(\mathbf{x}_2) \\ &= \alpha\mathbf{y}^* + (1-\alpha)\mathbf{y}^* \\ &= \mathbf{y}^*, \end{aligned}$$

where $0 < \alpha < 1$. Thus the inverse image is a convex set.

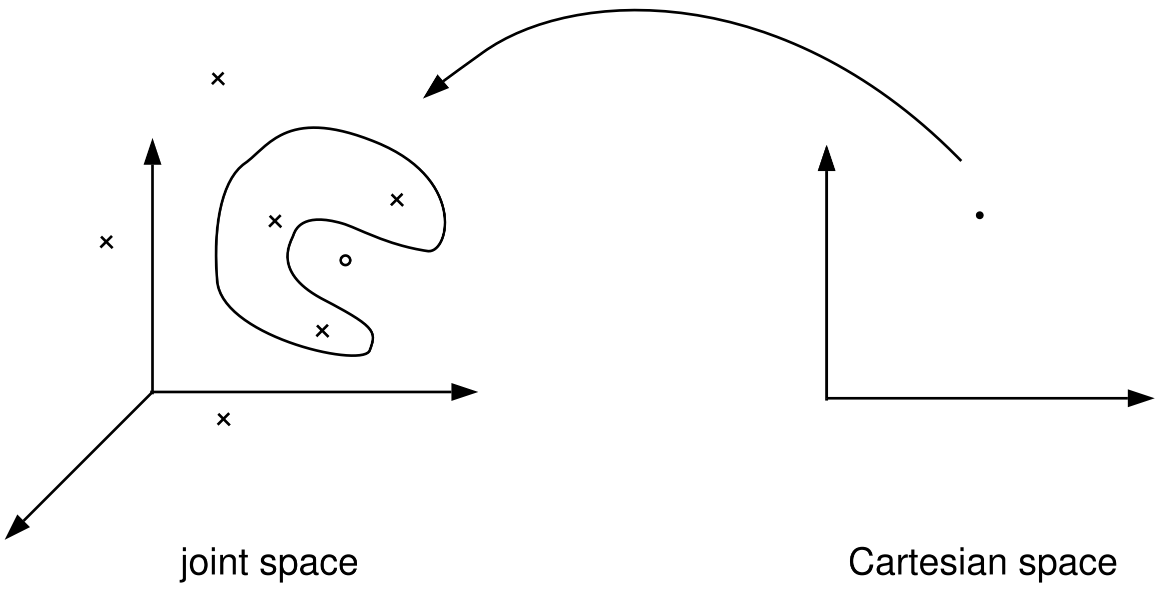

Figure 22: The convexity problem. The region on the left is the inverse image of the point on the right. The arrow represents the direction in which the mapping is learned by direct inverse modeling. The three points lying inside the inverse image are averaged by the learning procedure, yielding the vector represented by the small circle. This point is not in the inverse image, because the inverse image is not convex, and is therefore not a solution.

image (i.e., map into the same Cartesian position). The figure also shows the joint-space average of these two configurations (the dashed configuration in the figure). An average of two points lies on the straight line joining the points, thus the fact that the average configuration does not itself lie in the inverse image (i.e., does not map into the same Cartesian position) demonstrates that the inverse image is nonconvex. Interestingly, the Cartesian error observed in Figure 23 is essentially the same error as that observed in the corresponding position of the error vector field in Figure 21. This provides support for the assertion that the error vector field is due to the nonconvexities of the inverse kinematics.

Feedback error learning

Kawato, Furukawa and Suzuki (1987) have developed a direct approach to motor learning that avoids some of the difficulties associated with direct in-

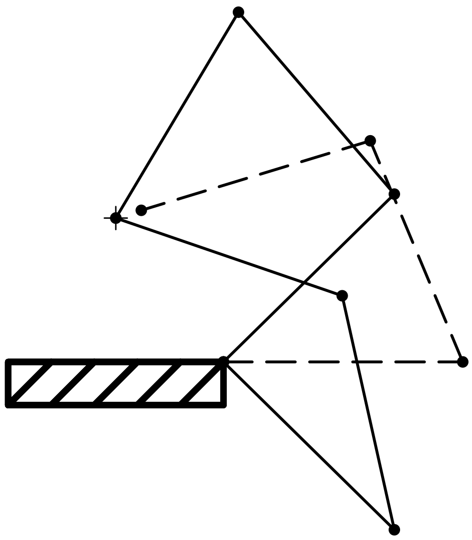

Figure 23: The nonconvexity of inverse kinematics. The dotted configuration is an average in joint space of the two solid configurations.

verse modeling. Their approach, known as *feedback error learning*, makes use of a feedback controller to guide the learning of the feedforward controller. Consider the composite feedback-feedforward control system discussed earlier (cf. Figure 9), in which the total control signal is the sum of the feedforward component and the feedback component:

$$\mathbf{u}[n] = \mathbf{u}_{ff}[n] + \mathbf{u}_{fb}[n].$$

In the context of a direct approach to motor learning, the signal $\mathbf{u}[n]$ is the target for learning the feedforward controller (cf. Figure 18). The error between the target and the feedforward control signal is $(\mathbf{u}[n] - \mathbf{u}_{ff}[n])$, which in the current case is simply $\mathbf{u}_{fb}[n]$. Thus an error for learning the feedforward controller can be provided by the feedback control signal (see Figure 24).

An important difference between feedback error learning and direct inverse modeling regards the signal used as the controller input. In direct inverse modeling the controller is trained "off-line;" that is, the input to the controller for the purposes of training is the actual plant output, not the desired plant output. For the controller to actually participate in the control process, it must receive the desired plant output as its input. The direct inverse modeling approach therefore requires a switching process—the desired plant output must be switched in for the purposes of control and the actual plant output must be switched in for the purposes of training. The feedback error learning approach

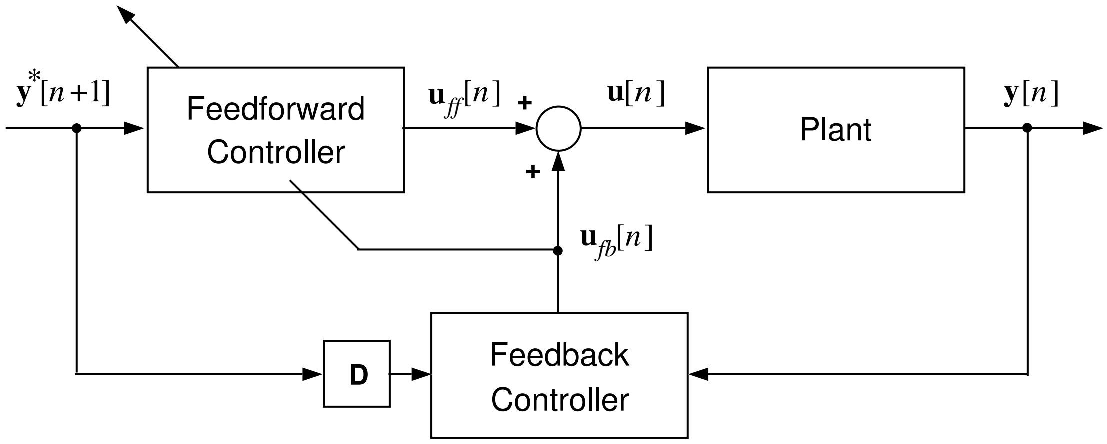

Figure 24: The feedback error learning approach to learning a feedforward controller. The feedback control signal is the error term for learning the feedforward controller.

provides a more elegant solution to this problem. In feedback error learning, the desired plant output is used for both control and training. The feedforward controller is trained "on-line;" that is, it is used as a controller while it is being trained. Although the training data that it receives—pairs of actual plant inputs and desired plant outputs—are not samples of the inverse dynamics of the plant, the system nonetheless converges to an inverse model of the plant because of the error-correcting properties of the feedback controller.

By utilizing a feedback controller, the feedback error learning approach also solves another problem associated with direct inverse modeling. Direct inverse modeling is not *goal directed*; that is, it is not sensitive to particular output goals (Jordan & Rosenbaum, 1989). This is seen by simply observing that the goal signal ($\mathbf{y}^*[n+1]$) does not appear in Figure 18. The learning process samples randomly in the control space, which may or may not yield a plant output near any particular goal. Even if a particular goal is specified before the learning begins, the direct inverse modeling procedure must search throughout the control space until an acceptable solution is found. In the feedback error learning approach, however, the feedback controller serves to guide the system to the correct region of the control space. By using a feedback controller, the system makes essential use of the error between the desired plant output and the actual plant output to guide the learning. This fact links the feedback error learning approach to the indirect approach to motor learning that we discuss in the following section. In the indirect approach, the learning algorithm is based directly on the output error.

Distal supervised learning

In this section we describe an indirect approach to motor learning known as distal supervised learning. Distal supervised learning avoids the nonconvexity problem and also avoids certain other problems associated with direct approaches to motor learning (Jordan, 1990; Jordan & Rumelhart, 1992). In distal supervised learning, the controller is learned indirectly, through the intermediary of a forward model of the plant. The forward model must itself be learned from observations of the inputs and outputs of the plant. The distal supervised learning approach is therefore composed of two interacting processes, one process in which the forward model is learned and another process in which the forward model is used in the learning of the controller.

In the case of a linear plant, the distal supervised learning approach is a cross between two techniques from adaptive control theory: indirect self-tuning control and indirect model reference adaptive control (Åström & Wittenmark, 1989). Let us begin by describing the basic idea of indirect self-tuning control, to provide some insight into how a forward model can be used as an intermediary in the learning of an inverse model. Consider once again the first-order example (Equation 41). Suppose that instead of learning an inverse model of the plant directly, the system first learns a forward model of the plant. We assume a parameterized forward plant model of the following form:

$$\hat{y}[n+1] = w_1 \hat{x}[n] + w_2 u[n], \tag{43}$$

where the weights w_1 and w_2 are unknown parameters. This equation is linear in the unknown parameters, thus the LMS algorithm is applicable. As in the previous section, we shift the time index backward by one time step to express the model in terms of the current plant output $y[n]$. This yields the wiring diagram shown in Figure 25. The forward model is an LMS processing unit with inputs $u[n-1]$ and $y[n-1]$, where $y[n-1]$ is the estimate $\hat{x}[n-1]$. The output of the LMS unit is the predicted plant output $\hat{y}[n]$ and the target for the learning algorithm is the actual plant output $y[n]$. By minimizing the *prediction error* $(y[n] - \hat{y}[n])$, the system adjusts the weights in the forward model.

Let us suppose that the learner has acquired a perfect forward model; that is, the predicted plant output is equal to the actual plant output for all states and all inputs. Equation 43 can now be inverted algebraically to provide an

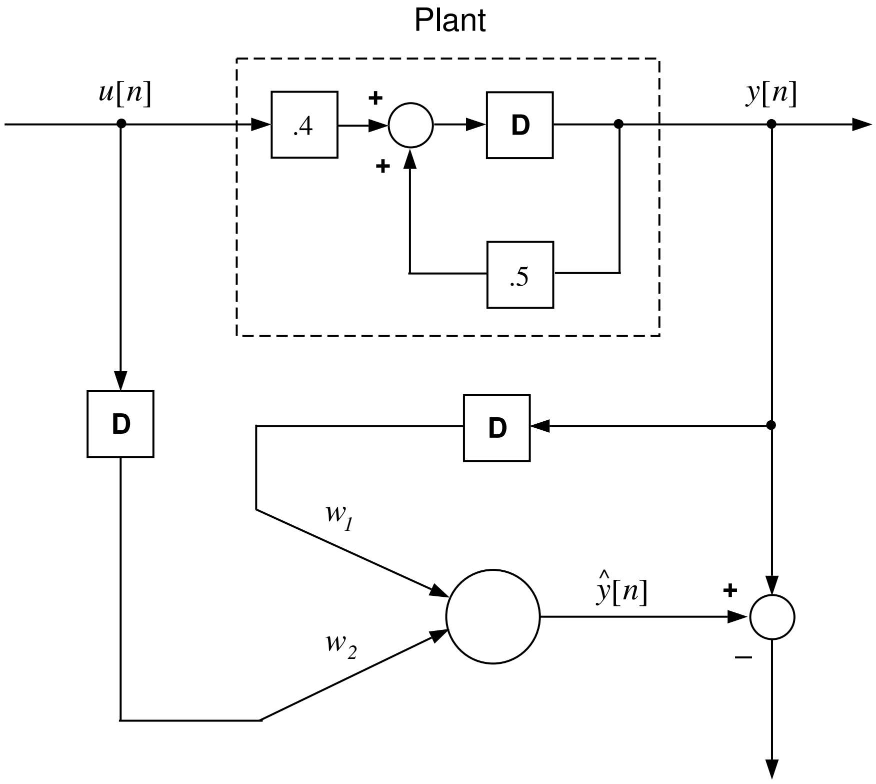

Figure 25: An example of the learning of the forward model in the distal supervised learning approach. An LMS processing unit is connected to the first order plant. The bias has been omitted for simplicity.

inverse model of the following form:

$$u[n] = -\frac{w_1}{w_2}\hat{x}[n] + \frac{1}{w_2}y^*[n+1]. \tag{44}$$

Note that in this equation, the weights of the forward model are being used to construct the inverse model. If the forward model is perfect, that is, if w_1 is equal to .5 and w_2 is equal to .4, then the inverse model is also perfect—the coefficients in Equation 44 are 1.25 and 2.5 (cf. Equation 15).

The nonlinear case

In the case of a linear plant, the differences between the direct approach to learning an inverse model and the indirect approach to learning an inverse model are relatively minor. Essentially, the choice is between performing the algebra first and then the learning, or the learning first and then the algebra. In the nonlinear case, however, the differences are much more salient. Indeed, it is not entirely clear how to proceed in the nonlinear case, given that nonlinear plant models are generally nonlinear in the parameters and are therefore difficult to invert algebraically.

To see how to proceed, let us reconsider the notion of an inverse model of the plant. Rather than defining an inverse model as a particular transformation from plant outputs to plant inputs, let us define an inverse model as any transformation that when placed in series with the plant yields the *identity transformation*. That is, an inverse model is any system that takes an input $\mathbf{y}^*[n+1]$ (at time n) and provides a control signal to the plant such that the plant output (at time $n+1$) is equal to $\mathbf{y}^*[n+1]$. This implicit definition of an inverse model recognizes that there may be more than one inverse model of the plant. Moreover, this definition suggests an alternative approach to training an inverse model. Suppose that we consider the controller and the plant together as a single composite system that transforms a desired plant output into an actual plant output. An indirect approach to training the controller is to train this composite system to be the identity transformation.

Stated in this manner, this indirect approach seems unrealizable, because learning algorithms require access to the internal structure of the system that they are training, and internal structure is precisely what is lacking in the case of the unknown physical plant. There is a way out, however, which involves using an internal forward model of the plant rather than the plant itself. This is the essence of the distal supervised learning approach, as illustrated diagrammatically in Figure 26. There are two interwoven processes depicted in the figure. One process involves the acquisition of an internal forward model of the plant. The forward model is a mapping from states and inputs to predicted plant outputs and it is trained using the *prediction error* $(\mathbf{y}[n] - \hat{\mathbf{y}}[n])$. The second process involves training the controller. This is accomplished in the following manner. The controller and the forward model are joined together and are treated as a single *composite learning system*. Using a nonlinear supervised learning algorithm, the composite system is trained to be an identity transformation. That is, the entire composite learning system (the system inside the dashed box in the figure) corresponds to the box labelled "Learner" in Figure 17. During this training process, the parameters in the forward model are held fixed. Thus the composite learning system is trained to be an identity transformation by a constrained learning process in which some of the parameters inside the system are held fixed. By allowing only the controller

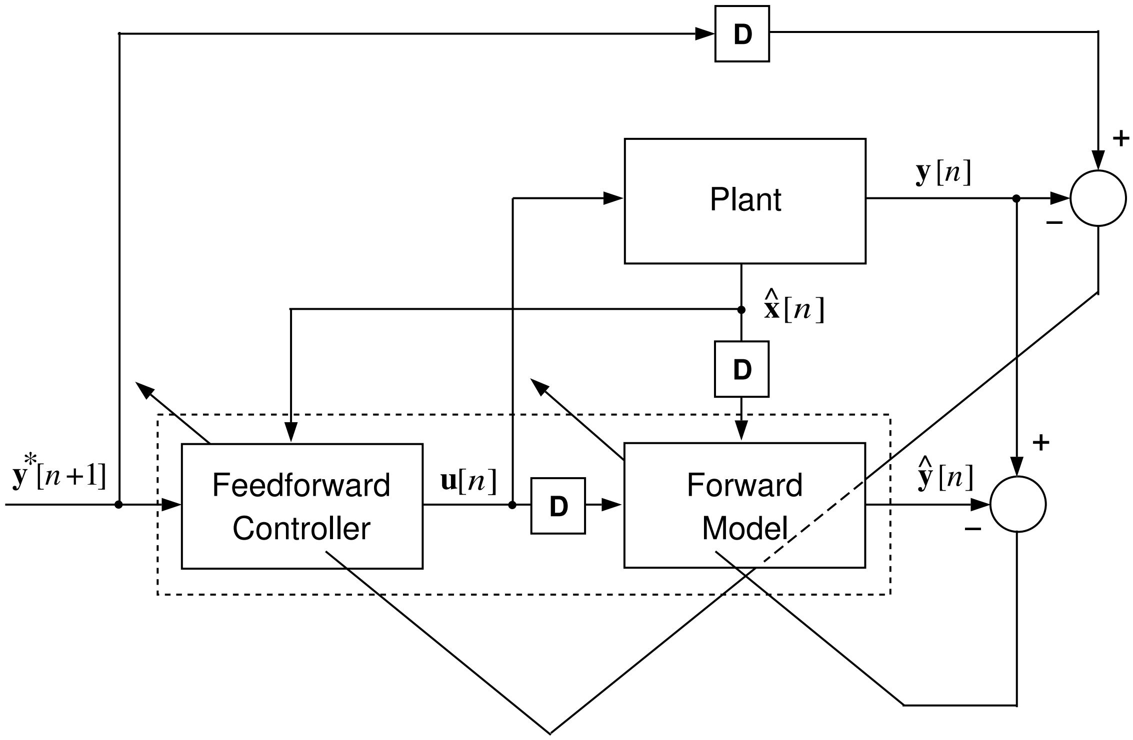

Figure 26: The distal supervised learning approach. The forward model is trained using the *prediction error* $(\mathbf{y}[n] - \hat{\mathbf{y}}[n])$. The subsystems in the dashed box constitute the *composite learning system*. This system is trained by using the *performance error* $(\mathbf{y}^*[n] - \mathbf{y}[n])$ and holding the forward model fixed. The state estimate $\hat{\mathbf{x}}[n]$ is assumed to be provided by an observer (not shown).

parameters to be altered, this process trains the controller indirectly.[11]

Let us consider the second component of this procedure in more detail. At any given time step, a desired plant output $\mathbf{y}^*[n+1]$ is provided to the controller and an action $\mathbf{u}[n]$ is generated. These signals are delayed by one time step before being fed to the learning algorithm, to allow the desired plant output to be compared with the actual plant output at the following time step. Thus the signals utilized by the learning algorithm (at time n) are the delayed desired output $\mathbf{y}^*[n]$ and the delayed action $\mathbf{u}[n-1]$. The delayed action is fed to the forward model, which produces an internal prediction ($\hat{\mathbf{y}}[n]$) of the actual plant output.[12] Let us assume, temporarily, that the forward model is a perfect model of the plant. In this case, the internal prediction ($\hat{\mathbf{y}}[n]$) is equal to the actual plant output ($\mathbf{y}[n]$). Thus the composite learning system, consisting of the controller and the forward model, maps an input $\mathbf{y}^*[n]$ into an output $\mathbf{y}[n]$. For the composite system to be an identity transformation these two signals must be equal. Thus the error used to train the composite system is the *performance error* ($\mathbf{y}^*[n]-\mathbf{y}[n]$). This is a sensible error term—it is the observed error in motor performance. That is, the learning algorithm trains the controller by correcting the error between the desired plant output and the actual plant output. Optimal performance is characterized by zero error. In contrast with the direct inverse modeling approach, the optimal least-squares solution for distal supervised learning is a solution in which the performance errors are zero.

Figure 27 shows the results of a simulation of the inverse kinematic learning problem for the planar arm. As is seen, the distal supervised learning approach avoids the nonconvexity problem and finds a particular inverse model of the arm kinematics. (For extensions to the case of learning multiple, context-sensitive, inverse models, see Jordan, 1990.)

Suppose finally that the forward model is imperfect. In this case, the

[11] It has been suggested (Miall, Weir, Wolpert, & Stein, in press) that the distal supervised learning approach requires using the backpropagation algorithm of Rumelhart, Hinton, and Williams (1986). This is not the case; indeed, a wide variety of supervised learning algorithms are applicable. The only requirement of the algorithm is that it obey an "architectural closure" property: a cascade of two instances of an architecture must itself be an instance of the architecture. This property is satisfied by a variety of algorithms, including the Boltzmann machine (Hinton & Sejnowski, 1986) and decision trees (Breiman, Friedman, Olshen, & Stone, 1984).

[12] The terminology of *efference copy* and *corollary discharge* may be helpful here (see, e.g., Gallistel, 1980). The control signal ($\mathbf{u}[n]$) is the efference, thus the path from the controller to the forward model is an efference copy. It is important to distinguish this efference copy from the internal prediction $\hat{\mathbf{y}}[n]$, which is the output of the forward model. (The literature on efference copy and corollary discharge has occasionally been ambiguous in this regard).

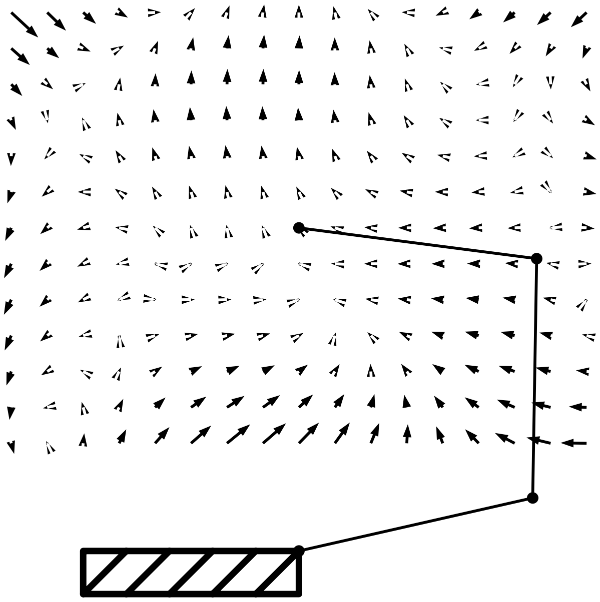

Figure 27: Near-asymptotic performance of distal supervised learning.

error between the desired output and the predicted output is the quantity $(\mathbf{y}^*[n] - \hat{\mathbf{y}}[n])$, the *predicted performance error*. Using this error, the best the system can do is to acquire a controller that is an inverse of the forward model. Because the forward model is inaccurate, the controller is inaccurate. However, the predicted performance error is not the only error available for training the composite learning system. Because the actual plant output $(\mathbf{y}[n])$ can still be measured after a learning trial, the true performance error $(\mathbf{y}^*[n] - \mathbf{y}[n])$ is still available for training the controller.[13] This implies that the output of the forward model can be discarded; the forward model is needed only for the structure that it provides as part of the composite learning system (see below for further clarification of this point). Moreover, for the purpose of providing internal structure to the learning algorithm, an exact forward model is not required. Roughly speaking, the forward model need only provide coarse

[13]This argument assumes that the subject actually performs the action. It is also possible to consider "mental practice" trials, in which the action is imagined but not performed (Minas, 1978). Learning through mental practice can occur by using the predicted performance error. This makes the empirically-testable prediction that the efficacity of mental practice should be closely tied to the accuracy of the underlying forward model (which can be assessed independently by measuring the subject's abilities at prediction or anticipation of errors).

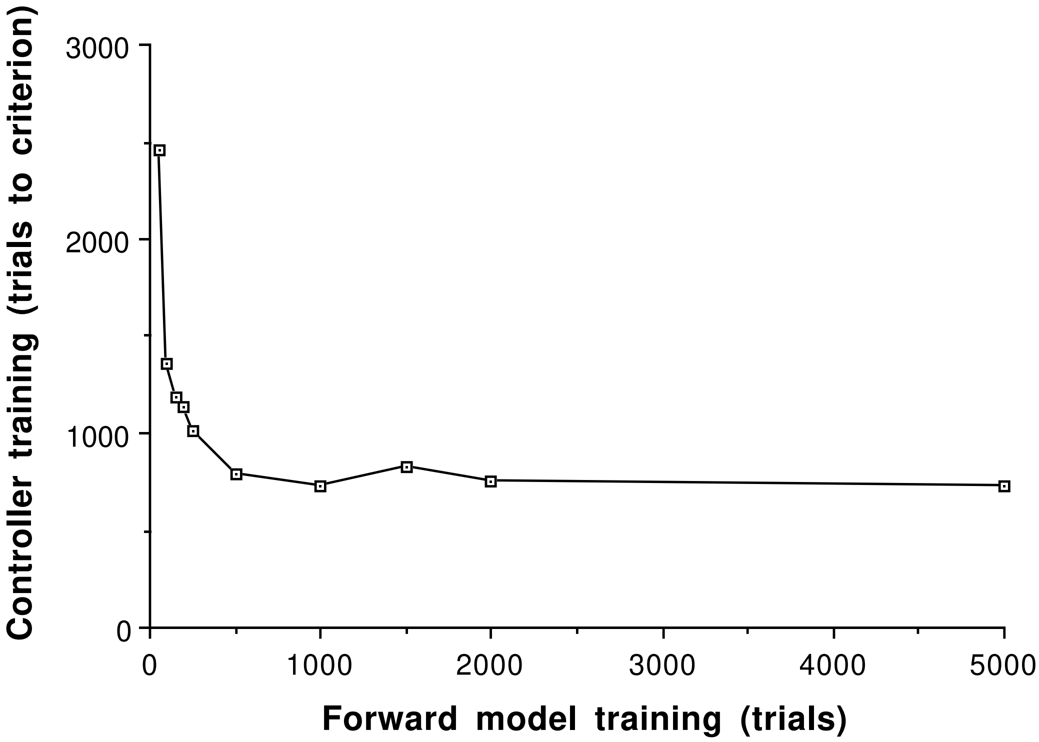

Figure 28: Number of trials required to train the controller to an error criterion of 0.001 as a function of the number of trials allocated to training the forward model.

information about how to improve the control signal based on the current performance error, not precise information about how to make the optimal correction. If the performance error is decreased to zero, then an accurate controller has been found, regardless of the path taken to find that controller. Thus an accurate controller can be learned even if the forward model is inaccurate. This point is illustrated in Figure 28, which shows the time required to train an accurate controller as a function of the time allocated to training the forward model. The accuracy of the forward model increases monotonically as a function of training time, but is still somewhat inaccurate after 5000 trials (Jordan & Rumelhart, 1992). Note that the time required to train the controller is rather insensitive to the accuracy of the forward model.

Example

Further insight into the distal supervised learning approach can be obtained

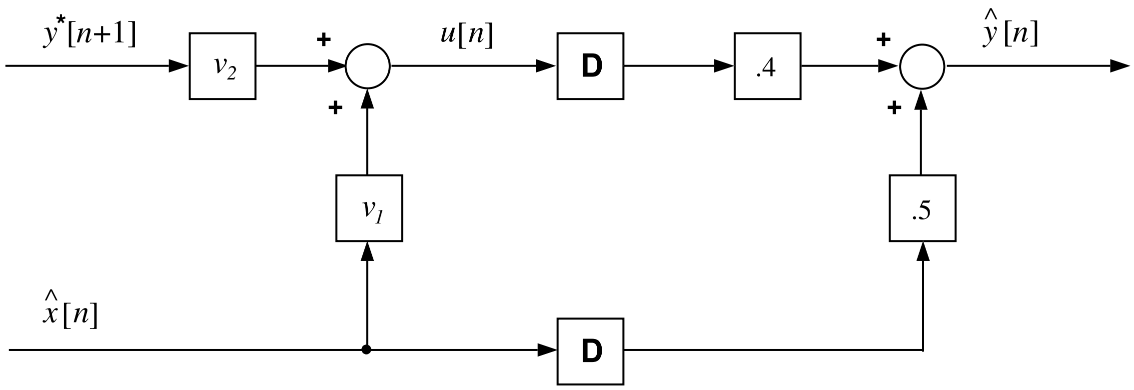

Figure 29: The composite learning system (the controller and the forward model) for the first-order example. The coefficients v_1 and v_2 are the unknown parameters in the controller. The coefficients .4 and .5 are the parameters in the forward model.

by reconsidering the linear problem described earlier. The composite learning system for this problem is shown in Figure 29. Note that we have assumed a perfect forward model—the parameters .5 and .4 in the forward model are those that describe the true plant. How might the performance error $(y^*[n] - y[n])$ be used to adjust the parameters v_1 and v_2 in the controller? Suppose that the performance error is positive; that is, suppose that $y^*[n]$ is greater than $y[n]$. Because of the positive coefficient (.4) that links $u[n-1]$ and $y[n]$, to increase $y[n]$ it suffices to increase $u[n-1]$. To increase $u[n-1]$ it is necessary to adjust v_1 and v_2 appropriately. In particular, v_1 should increase if $\hat{x}[n-1]$ is positive and decrease otherwise; similarly, v_2 should increase if $y^*[n]$ is positive and decrease otherwise. This algorithm can be summarized in an LMS-type update rule:

$$\Delta v_i = \mu \ \mathrm{sgn}(w_2) \ (y^*[n] - y[n]) z_i[n-1],$$

where $z_1[n-1] \equiv \hat{x}[n-1]$, $z_2[n-1] \equiv y^*[n]$, and $\mathrm{sgn}(w_2)$ denotes the sign of w_2 (negative or positive), where w_2 is the forward model's estimate of the coefficient linking $u[n-1]$ and $y[n]$ (cf. Equation 43).

Note the role of the forward model in this learning process. The forward model is required in order to provide the sign of the parameter w_2 (the coefficient linking $u[n-1]$ and $\hat{y}[n]$). The parameter w_1 (that linking $\hat{x}[n-1]$ and $\hat{y}[n]$) is needed only during the learning of the forward model to ensure that the correct sign is obtained for w_2. A very inaccurate forward model suffices for learning the controller—only the sign of w_2 needs to be correct. Moreover, it is likely that the forward model and the controller can be learned simultaneously, because the appropriate sign for w_2 will probably be discovered early in the learning process.

Reference Models

Throughout this chapter we have characterized controllers as systems that invert the plant dynamics. For example, a predictive controller was characterized as an inverse model of the plant—a system that maps desired plant outputs into the corresponding plant inputs. This mathematical ideal, however, is not necessarily realizable in all situations. One common difficulty arises from the presence of constraints on the magnitudes of the control signals. An ideal inverse model moves the plant to an arbitrary state in a small number of time steps, the number of steps depending on the order of the plant. If the current state and the desired state are far apart, an inverse model may require large control signals, signals that the physical actuators may not be able to provide. Moreover, in the case of feedback control, large control signals correspond to high gains, which may compromise closed-loop stability. A second difficulty is that the inverses of certain dynamical systems, known as "nonminimum phase" systems, are unstable (Åström & Wittenmark, 1984). Implementing an unstable inverse model is clearly impratical, thus another form of predictive control must be sought for such systems.[14]

As these considerations suggest, realistic control systems generally embody a compromise between a variety of constraints, including performance, stability, bounds on control magnitudes, and robustness to disturbances. One way to quantify such compromises is through the use of a *reference model*. A reference model is an explicit specification of the desired input-output behavior of the control system. A simple version of this idea was present in the previous section when we noted that an inverse model can be defined implicitly as any system that can be cascaded with the plant to yield the identity transformation. From the current perspective, the identity transformation is the simplest and most stringent reference model. An identity reference model requires, for example, that the control system respond to a sudden increment in the controller input with a sudden increment in the plant output. A more forgiving

[14]A discussion of nonminimum phase dynamics is beyond the scope of this chapter, but an example would perhaps be useful. The system $y[n+1] = .5y[n] + .4u[n] - .5u[n-1]$ is a nonminimum phase system. Solving for $u[n]$ yields the inverse model $u[n] = -1.25y[n] + 2.5y^*[n+1] + 1.25u[n-1]$, which is an unstable dynamical system, due to the coefficient of 1.25 that links successive values of u.

reference model would allow the plant output to rise more smoothly to the desired value. Allowing smoother changes in the plant output allows the control signals to be of smaller magnitude.

Although reference models can be specified in a number of ways, for example as a table of input-output pairs, the most common approach is to specify the reference model as a dynamical system. The input to the reference model is the *reference signal*, which we now denote as $\mathbf{r}[n]$, to distinguish it from the reference model output, which we denote as $\mathbf{y}^*[n]$. The reference signal is also the controller input. This distinction between the reference signal and the desired plant output is a useful one. In a model of speech production, for example, it might be desirable to treat the controller input as a linguistic "intention" to produce a given phoneme. The phoneme may be specified in a symbolic linguistic code that has no intrinsic articulatory or acoustic interpretation. The linguistic intention $\mathbf{r}[n]$ would be tied to its articulatory realization $\mathbf{u}[n]$ through the controller and also tied to its (desired) acoustic realization $\mathbf{y}^*[n]$ through the reference model. (The actual acoustic realization $\mathbf{y}[n]$ would of course be tied to the articulatory realization $\mathbf{u}[n]$ through the plant.)

Example

Let us design a model-reference controller for the first-order plant discussed earlier. We use the following reference model:

$$y^*[n+2] = s_1 y^*[n+1] + s_2 y^*[n] + r[n], \tag{45}$$

where $r[n]$ is the reference signal. This reference model is a second-order difference equation in which the constant coefficients s_1 and s_2 are chosen to give a desired dynamical response to particular kinds of inputs. For example, s_1 and s_2 might be determined by specifying a particular desired response to a step input in the reference signal. Let us write the plant dynamical equation at time $n+2$:

$$y[n+2] = .5y[n+1] + .4u[n+1].$$

We match up the terms on the right-hand sides of both equations and obtain the following control law:

$$u[n] = (\frac{s_1 - .5}{.4})y[n] + \frac{s_2}{.4}y[n-1] + \frac{1}{.4}r[n-1]. \tag{46}$$

Figure 30 shows the resulting model-reference control system. This control system responds to reference signals ($r[n]$) in exactly the same way as the reference model in Equation 45.

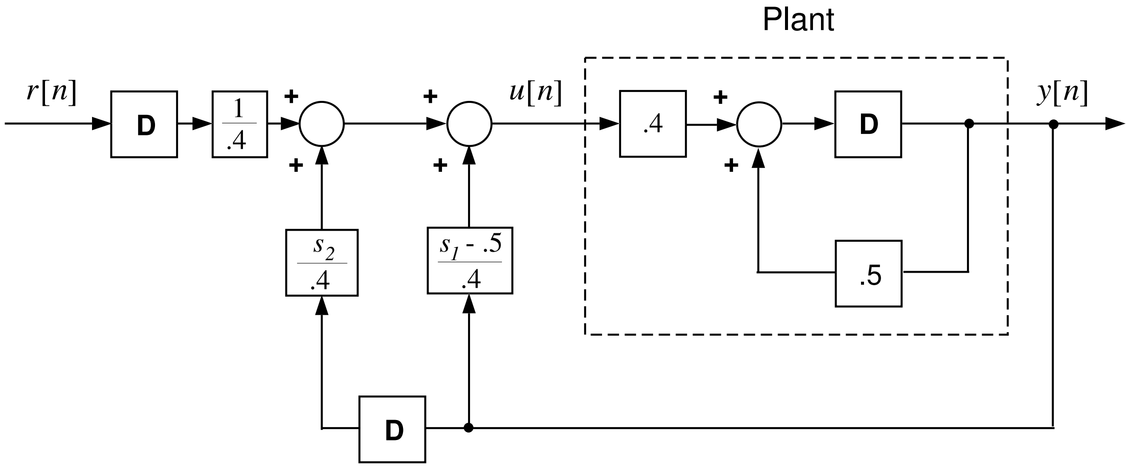

Figure 30: The model-reference control system for the first-order example.

Note that the reference model itself does not appear explicitly in Figure 30. This is commonly the case in model-reference control; the reference model often exists only in the mind of the person designing or analyzing the control system. It serves as a guide for obtaining the controller, but it is not implemented as such as part of the control system. On the other hand, it is also possible to design model-reference control systems in which the reference model does appear explicitly in the control system. The procedure is as follows. Suppose that an inverse model for a plant has already been designed. We know that a cascade of the inverse model and the plant yields the identity transformation. Thus a cascade of the reference model, the inverse model, and the plant yields a composite system that is itself equivalent to the reference model. The controller in this system is the cascade of the reference model and the inverse model. At first glance this approach would appear to yield little net gain because it involves implementing an inverse model. Note, however, that despite the presence of the inverse model, this approach provides a solution to the problem of excessively large control signals. Because the inverse model lies after the reference model in the control chain, its input is smoother than the reference model input, thus it will not be required to generate large control signals.

Turning now to the use of reference models in learning systems, it should be clear that the distal supervised learning approach can be combined with the use of reference models. In the section on distal supervised learning, we described how an inverse plant model could be learned by using the identity transformation as a reference model for the controller and the forward model. Clearly, the identity transformation can be replaced by any other reference model and the same approach can be used. The reference model can be thought of as a source of input-output pairs for training the controller and the forward model (the composite learning system), much as the plant is a source of input-output pairs for the training of the forward model. The distal supervised learning training procedure finds a controller that can be cascaded with the plant such that the resulting composite control system behaves as specified by the reference model.

It is important to distinguish clearly between forward and inverse models on the one hand and reference models on the other. Forward models and inverse models are internal models of the plant. They model the relationship between plant inputs and plant outputs. A reference model, on the other hand, is a specification of the desired behavior of the control system, from the controller input to the plant output. The signal that intervenes between the controller and the plant (the plant input) plays no role in a reference model specification. Indeed, the same reference model may be appropriate for plants having different numbers of control inputs or different numbers of states. A second important difference is that forward models and inverse models are actual dynamical systems, implemented as internal models "inside" the organism. A reference model need not be implemented as an actual dynamical system; it may serve only as a guide for the design or the analysis of a control system. Alternatively, the reference model may be an actual dynamical system, but it may be "outside" of the organism and known only by its inputs and outputs. For example, the problem of learning by imitation can be treated as the problem of learning from an external reference model. The reference model provides only the desired behavior; it does not provide the control signals needed to perform the desired behavior.

Conclusions

If there is any theme that unites the various techniques that we have discussed, it is the important role of internal dynamical models in control systems. The two varieties of internal models—inverse models and forward models—play complementary roles in the implementation of sophisticated control strategies. Inverse models are the basic module for predictive control, allowing the system to precompute an appropriate control signal based on a desired plant output. Forward models have several roles: they provide an alternative implementation of feedforward controllers, they can be used to anticipate and cancel delayed feedback, they are the basic building block in dynamical state estimation, and they play an essential role in indirect approaches to motor learning. In general, internal models provide capabilities for prediction, control, and error correction that allow the system to cope with difficult nonlinear control problems.

It is important to emphasize that an internal model is not necessarily a detailed model, or even an accurate model, of the dynamics of the controlled system. In many cases, approximate knowledge of the plant dynamics can be used to move a system in the "right direction." An inaccurate inverse model can provide an initial push that is corrected by a feedback controller. An inaccurate forward model can be used to learn an accurate controller. Inaccurate forward models can also be used to provide partial cancellation of delayed feedback, and to provide rough estimates of the state of the plant. The general rule is that partial knowledge is better than no knowledge, if used appropriately.

These observations would seem to be particularly relevant to human motor control. The wide variety of external dynamical systems with which humans interact, the constraints on the control system due to delays and limitations on force and torque generation, and the time-varying nature of the musculoskeletal plant all suggest an important role for internal models in biological motor control. Moreover, the complexity of the systems involved, as well as the unobservability of certain aspects of the environmental dynamics, make it likely that the motor control system must make do with approximations. It is of great interest to characterize the nature of such approximations. Although approximate internal models can often be used effectively, there are deep theoretical issues involved in characterizing how much inaccuracy can be tolerated in various control system components. The literature on control theory is replete with examples in which inaccuracies lead to instabilities, if care is not taken in the control system design. As theories of biological motor control increase in sophistication, these issues will be of increasing relevance.

References

Anderson, B. D. O., & Moore, J. B. (1979). *Optimal Filtering.* Englewood Cliffs, NJ: Prentice-Hall.

Åström, K. J., & Wittenmark, B. (1984). *Computer controlled systems: Theory and design.* Englewood Cliffs, NJ: Prentice-Hall.

Åström, K. J., & Wittenmark, B. (1989). *Adaptive Control.* Reading, MA: Addison-Wesley.

Atkeson, C. G., & Reinkensmeyer, D. J. (1988). Using associative content-addressable memories to control robots. *IEEE Conference on Decision and Control*. San Francisco, CA.

Atkeson, C. G. (1990). Using local models to control movement. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 2*, pp. 316-324. San Mateo, CA: Morgan Kaufmann.

Bernstein, N. (1967). *The coordination and regulation of movements*. London: Pergamon.

Breiman, L., Friedman, J.H., Olshen, R.A., & Stone, C.J. (1984) *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group.

Carlton, L. G. (1981). Processing visual feedback information for movement control. *Journal of Experimental Psychology: Human Perception and Performance*, *7*, 1019-1030.

Duda, R. O., & Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. New York: Wiley.

Friedman, J.H. (1990). Multivariate adaptive regression splines. *The Annals of Statistics, 19*, 1-141.

Galliana, H. L., & Outerbridge, J. S. (1984). A bilateral model for central neural pathways in the vestibuloocular reflex. *Journal of Neurophysiology*, *51*, 210-241.

Gallistel, C. R. (1980). *The Organization of Action*. Hillsdale, NJ: Erlbaum.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, *4*, 1-59.

Goodwin, G. C., & Sin, K. S. (1984). *Adaptive filtering prediction and control*. Englewood Cliffs, NJ: Prentice-Hall.

Haken, H., Kelso, J. A. S., & Bunz, H. (1985). A theoretical model of phase transitions in human hand movements. *Biological Cybernetics*, *51*, 347-356.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, *40*, 185-234.

Hinton, G. E., & Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing*: *Volume 1*, 282-317. Cambridge, MA: MIT Press.

Hogan, N. (1984). An organising principle for a class of voluntary movements. *Journal of Neuroscience*, *4*, 2745-2754.

Hollerbach, J. M. (1982). Computers, brains, and the control of movement. *Trends in Neuroscience*, *5*, 189-193.

Jordan, M. I., & Rosenbaum, D. A. (1989). Action. In M. I. Posner (Ed.), *Foundations of Cognitive Science*. Cambridge, MA: MIT Press.

Jordan, M.I. (1990). Motor learning and the degrees of freedom problem. In M. Jeannerod (Ed.), *Attention and Performance, XIII*. Hillsdale, NJ: Erlbaum.

Jordan, M. I. (1992). Constrained supervised learning. *Journal of Mathematical Psychology*, *36*, 396-425.

Jordan, M. I., & Jacobs, R. A. (1992). Hierarchies of adaptive experts. In J. Moody, S. Hanson, & R. Lippmann (Eds.), *Advances in Neural Information Processing Systems 4*, pp. 985-993. San Mateo, CA: Morgan Kaufmann.

Jordan, M. I., & Rumelhart, D. E. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive Science*, *16*, 307-354.

Kawato, M. (1990). Computational schemes and neural network models for formation and control of multijoint arm trajectory. In W. T. Miller, III, R. S. Sutton, & P. J. Werbos (Eds.), *Neural Networks for Control*. Cambridge: MIT Press.

Kawato, M., Furukawa, K., & Suzuki, R. (1987). A hierarchical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, *57*, 169-185.

Keele, S., & Posner, M. (1968). Processing of visual feedback in rapid movements. *Journal of Experimental Psychology*, *77*, 155-158.

Kelso, J. A. S. (1986). Pattern formation in speech and limb movements involving many degrees of freedom. *Experimental Brain Research*, *15*, 105-128.

Koh, K., & Meyer, D. E. (1991). Function learning: Induction of continuous stimulus-response relations. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *17*, 811-837.

Kuperstein, M. (1988). Neural model of adaptive hand-eye coordination for single postures. *Science*, *239*, 1308-1311.

Lindblom, B., Lubker, J., & Gay, T. (1979). Formant frequencies of some fixed-mandible vowels and a model of speech motor programming by predictive simulation. *Journal of Phonetics*, *7*, 147-161.

Miall, R. C., Weir, D. J., Wolpert, D. M., & Stein, J. F. (in press). Is the cerebellum a Smith predictor? *Journal of Motor Behavior*.

Miller, W. T. (1987). Sensor-based control of robotic manipulators using a general learning algorithm. *IEEE Journal of Robotics and Automation*, *3*, 157-165.

Minas, S. C. (1978). Mental practice of a complex perceptual motor skill. *Journal of Human Movement Studies*, *4*, 102-107.

Misawa, E. A., & Hedrick. J. K. (1989). Nonlinear observers: A state-of-the-art survey. *ASME Journal of Dynamic Systems, Measurement, and Control*, *111*, 344-352.

Poggio, T., & Girosi, F. (1990). Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, *247*, 978-982.

Robinson, D. A. (1981). The use of control system analysis in the neurophysiology of eye movements. *Annual Review of Neuroscience*, *4*, 463-503.

Rosenblatt, F. (1962). *Principles of neurodynamics*. New York: Spartan.

Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing*: *Volume 1*, 318-363. Cambridge, MA: MIT Press.

Saltzman, E. L. (1979). Levels of sensorimotor representation. *Journal of Mathematical Psychology*, *20*, 91-163.

Schmidt, R. A. (1975). A schema theory of discrete motor skill learning. *Psychological Review*, *82*, 225-260.

Specht, D. F. (1991). A general regression neural network. *IEEE Transactions on Neural Networks*, *2*, 568-576.

Turvey, M. T., Shaw, R. E., & Mace, W. (1978). Issues in the theory of action: Degrees of freedom, coordinative structures and coalitions. In J. Requin (Ed.), *Attention and Performance, VII.* Hillsdale, NJ: Erlbaum.

Wahba, G. (1990). *Spline models for observational data.* Philadelphia, PA: SIAM.

Werbos, P. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* Unpublished doctoral dissertation, Harvard University.

Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record*, *Part 4*, 96-104.

Widrow, B., & Stearns, S. D. (1985). *Adaptive signal processing.* Englewood Cliffs, NJ: Prentice-Hall.