Chapter 4

Feedback-error control

4.1 Introduction

This chapter explains the feedback-error (FBE) control scheme originally described by Kawato [60, 87, 38]. FBE is a widely used neural network based controller which learns to generate the correct control inputs to drive a system's state variables along a desired trajectory. It has been successfully used in a number of control applications, including robot arm control [87], an automatic car braking system [92], and learning nonlinear thruster dynamics to control an underwater robotic vehicle [131].

FBE was originally intended by Kawato to be an abstract model of some of the low level motor areas of the brain, although in the engineering context it has outgrown its biological relevance. The motivation for studying FBE here is that it is a precursor to the FOX controller, with some of the same operational properties (although FOX is far more flexible).

4.2 The basics

This section describes the essentials of FBE control. The arguments are based around a continuous-time system, but identical arguments apply to discrete-time systems. The typical FBE system is shown in figure 4.1. The system being controlled, F, obeys the dynamical equation:

$$\dot{y} \quad = \quad F(y, x) \tag{4.1}$$

In general no restrictions are placed on the function F, but this work is mostly concerned with physical mechanical systems such as robotic manipulators. The vector $y(t)$ is the controlled system's state vari-

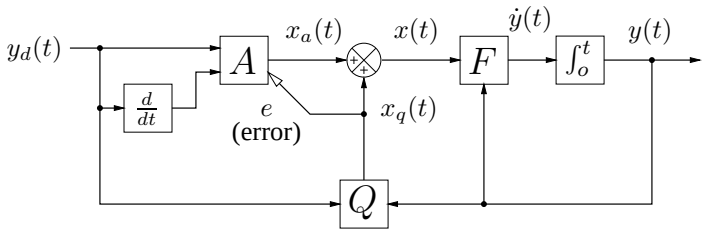

Figure 4.1: *The feedback-error controller.*

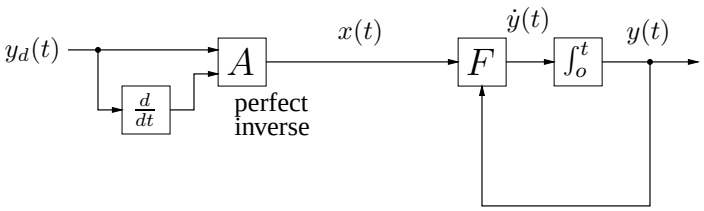

Figure 4.2: *The feed-forward path.*

ables (e.g. positions and velocities) and $x(t)$ is the vector of control variables (e.g. forces or torques). The vector $y_d(t)$ is the *desired* value of $y(t)$, supplied by some external agent. The purpose of the controller, of course, is to make sure $y(t)$ follows $y_d(t)$.

The controller has two main components. First, there is a feed-forward path through A, which is a stateless trainable module (such as a CMAC) that implements

$$x_a = A(y_d, \dot{y}_d) \tag{4.2}$$

Second, there is a feedback path through Q, which is a stateless fixed controller that implements

$$x_q = Q(y_d, y) \tag{4.3}$$

To understand figure 4.1 the feed-forward and feedback paths are examined separately. The convention used is that control operation starts at time $t = 0$.

4.3 The feed-forward path

Figure 4.2 shows the feed-forward path of the controller. In this figure

$$\dot{y} = F\Big(y, A(y_d, \dot{y}_d)\Big) \tag{4.4}$$

Now consider this: if $y(0) = y_d(0)$ then it is possible for A to output a value of x which makes $\dot{y}(0) = \dot{y}_d(0)$, assuming that such a value exists (because A is given the value of y that is also given to F). Extending this argument forward in time, if A is a *perfect* inverse model of the dynamic system it can maintain $y(t) = y_d(t)$ indefinitely for $t > 0$. In other words, it can control the system perfectly by itself. For this to work the *initial* desired and actual states are required to be the same ($y(0) = y_d(0)$) because A has no way of knowing the value of $y(t)$ other than the assumption that $y(t)$ *always* equals $y_d(t)$. If $\dot{y}_d$ is not available then the feed-forward controller will not work, because y_d can't supply information about both the current and desired states of the dynamic system.

A purely feed-forward controller is not practical for two reasons. First, the controller A can never be made perfect, and second, any noise or disturbance in the system will cause y and y_d to diverge.

4.4 The feedback path

Figure 4.3 shows the feedback path of the controller, which implements

$$x_q = Q(y_d, y) \tag{4.5}$$

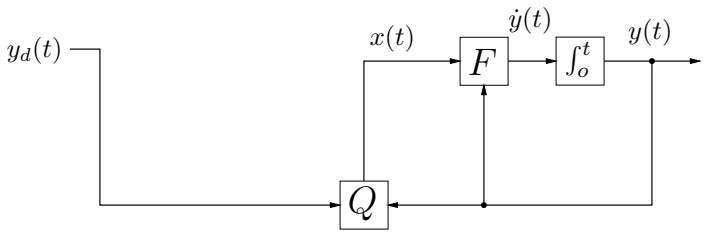

Figure 4.3: *The feedback path.*

The form of Q is arbitrary, although it is usually something like a simple proportional-plus-derivative controller. If Q is chosen well enough then y will tend towards y_d over time. With only the feedback path theoretical perfection can not be achieved (as with the feed-forward path), but practical, stable and robust control is possible.

4.5 Feed-forward and feedback together

To get the best of both types of control the feed-forward and feedback paths can be put together as shown in figure 4.1. The feed-forward path provides instant control signals (x_a) to drive y along the trajectory y_d. The feedback path stabilizes the system by compensating for any imperfections in A or any disturbances. Q provides control signals (x_q) to restore y to the set-point y_d when the two diverge.

If the dynamic system is nonlinear then a linear Q can only be chosen optimally for one operating point. It will be assumed that F is such that a Q can be found that works for the entire state space (albeit with varying degrees of efficiency). Systems which meet this constraint include robotic manipulators. This assumption will be important when FBE training is considered.

4.6 Feedback-error training

In general it is hard to determine the function A analytically for a dynamic system. Even if F is known exactly (an unlikely prospect for all but the simplest systems), computing an inverse has many problems. Thus A is made adaptive (i.e. parameterized by some weight vector w). In practice A is usually some kind of neural network.

The innovation of feedback-error control is that the training (error) signal e given to A is just the feedback signal x_q (hence the name "feedback-error"). Here is the reasoning behind this: if $x_q = 0$ then y will normally equal y_d. In this case the system is perfectly controlled, so the desired error signal is also zero because no adjustment to A is necessary. If $x_q \neq 0$ then the feedback controller is trying to correct the state y to match y_d. For better control, this correction should have been applied by the feed-forward controller some time earlier. Thus x_q is given as an error signal to A so that at this point in the future A will apply a better signal x_a and Q will have less to do (i.e. x_q will be smaller). The signal x_q has the interpretation "this is what x_a should have been". This relies on the generalization properties (local or global) of A. A is trained to minimize x_q and thereby become the inverse of F.

During training there is a transfer of control influence from the "unskilled" feedback controller loop (x_q) to the "skilled" feed-forward controller (x_a). Note that during training, x_a is moved in the *direction* of e (i.e. the error signal is not a target). Now, if

$$x_a = A(y_d, \dot{y}_d, w) \tag{4.6}$$

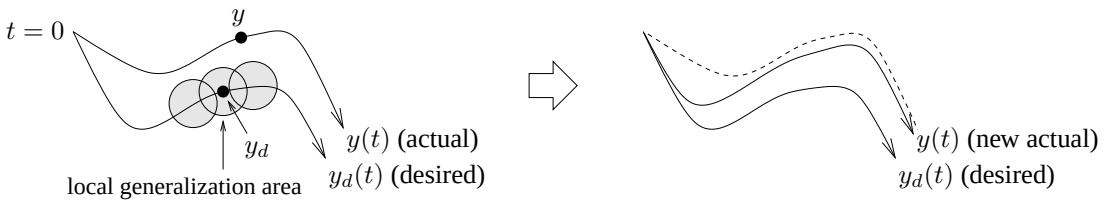

Figure 4.4: *How FBE controller training works.*

where w is a vector of weights controlling A, and if it is assumed that $e = \mathrm{d}E/\mathrm{d}x_a$ (where E is some hypothetical global scalar error) then the gradient descent training algorithm is

$$\dot{w} \quad = \quad \alpha \left[\frac{\mathrm{d}\, x_a}{\mathrm{d}\, w} \right]^T \cdot e \tag{4.7}$$

Here α is a learning rate constant. Note that for the controller to be effective over a certain volume of the state space, the states experienced during training must cover that volume.

4.7 Why it works

Why is the FBE training scheme successful? Figure 4.4 shows how a possible state-space trajectory for y changes during training. Assume that the controller is being repeatedly trained for the same trajectory $y_d(t)$, and that $y(0) = y_d(0)$. Assume also that A implements local generalization so that the training signal $e(t)$ affects a small region around the input point in A (this assumption is not strictly necessary but it will clarify the argument). During each training cycle the same trajectory $(y_d(t), \dot{y}_d(t))$ will be observed by A, so in effect a function of one time input is being trained.

In the early training cycles, y and y_d will start off the same and then gradually diverge (though not too much because of the feedback path). Now, if $y \neq y_d$ near the start of the trajectory, a nonzero error signal x_q will be associated with y_d. When this same desired trajectory is observed in subsequent training runs, the improved x_a will be asserted by A, and the divergence of y from y_d will be reduced. The argument proceeds by induction: as each part of the trajectory $y(t)$ converges to $y_d(t)$, the conditions are set for the convergence of later parts of $y(t)$. Thus the entire system will eventually converge. This argument can be extended to the general case where y_d is free roaming and not going through identical cycles.

4.8 Some other FBE features

4.8.1 Step changes in y_d

Step changes in the desired state y_d will not be handled well with this system, as then $\dot{y}_d$ would be infinite. Instead a trajectory with a smooth first derivative must be computed and passed to the controller.

4.8.2 Unskilled feedback

When the controller is fully trained, feedback has little effect on the control performance. This is acceptable in a "perfect" system, because then y always equals y_d. But in a real system, noise and other disturbances will affect y so that $y \neq y_d$. This is a problem, because the feed-forward controller does

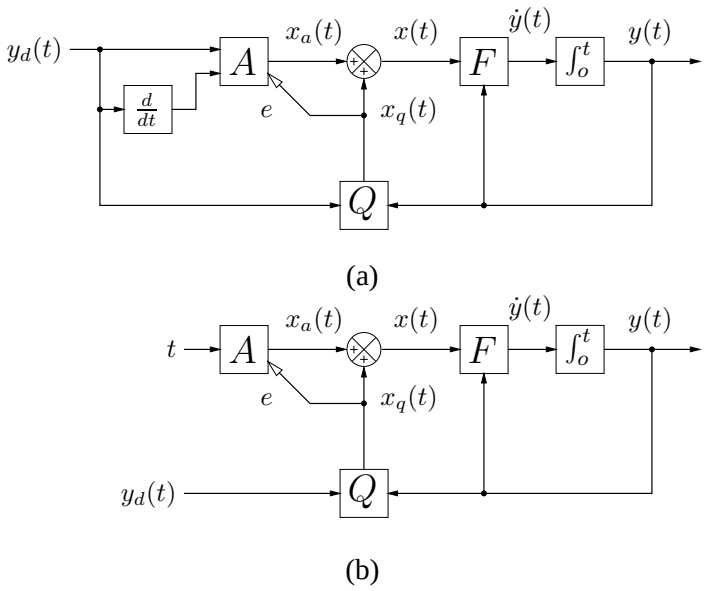

Figure 4.5: *(a) Standard feedback-error (FBE) control. (b) The inputs to A can be replaced by the time t if y_d is always the same one-dimensional trajectory.*

not get any information about the true state of y, it just assumes $y = y_d$. Thus any discrepancies must be dealt with by the "unskilled" feedback controller. So, although the feed-forward controller will cope with the system nonlinearity, noise performance and disturbance rejection are dependent on Q alone.

4.9 An example

It will now be demonstrated how FBE performs on a simple second order system, which corresponds to a unit mass at position p being driven along a straight line by a force x:

$$y = \begin{bmatrix} p \\ \dot{p} \end{bmatrix} \quad , \quad \dot{y} = \begin{bmatrix} \dot{p} \\ \ddot{p} \end{bmatrix} = F(y, x) = \begin{bmatrix} \dot{p} \\ x \end{bmatrix} \tag{4.8}$$

Q is a proportional-plus-derivative feedback controller that tries to get the position p and velocity $\dot{p}$ equal to their reference values:

$$x_q = k_p(p_d - p) + k_v(\dot{p}_d - \dot{p}) \tag{4.9}$$

where k_p and k_v are gain constants. The adaptive controller A is a CMAC neural network.

First a simplification will be made to the standard FBE controller shown in figure 4.5a. If the FBE system is repeatedly trained for the same reference trajectory $y_d(t)$ then A's input $[y_d(t), \dot{y}_d(t)]$ will describe a one-dimensional path parameterized by time t. Thus the inputs to A can be replaced by the time t to get effectively the same system, as shown in figure 4.5b. This simplification will help the following discussion.

Figure 4.6 shows the result of 8000 training iterations with a continuous reference trajectory that has $y_d(0) = y(0)$. The system parameters are given in the figure caption. Note that one iteration is a single run through the entire reference trajectory. The FBE controller is successful in this case—the

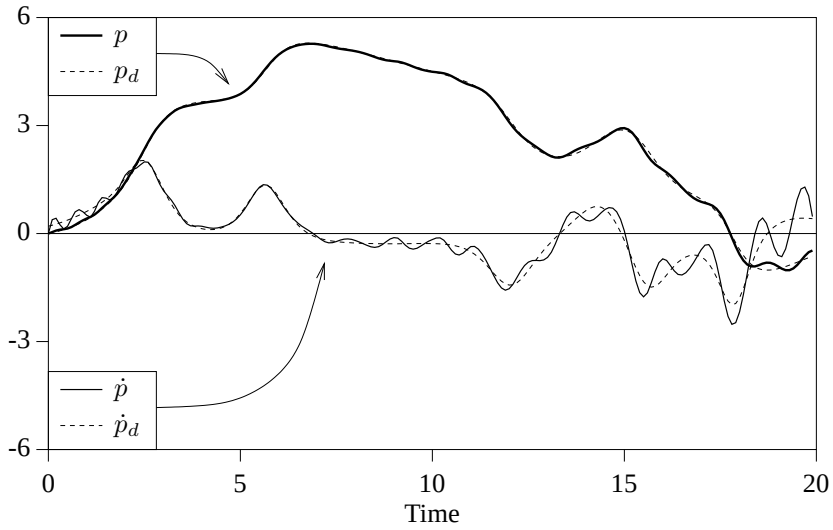

Figure 4.6: *Performance of the FBE second order example system with a continuous reference trajectory that has $y_d(0) = y(0)$. Convergence has almost been achieved after 8000 training iterations. A 200 time step Euler simulation was used with a step size of 0.1. The controller A was a single input CMAC with 200 input quantization levels and $n_a = 10$. The CMAC was trained with a learning rate of 0.05. The feedback controller had $k_p = k_v = 0.5$.*

position p and velocity $\dot{p}$ have mostly converged to their references (although the velocity still has some unconverged oscillations about its reference).

As has already been explained, FBE does not achieve convergence along the entire trajectory at once. Instead, earlier times converge before later times. This is demonstrated in figure 4.7, which shows the same system as in figure 4.6 except the graph was generated after only 600 training iterations. The position and velocity track their reference values for early times, but diverge for later times.

4.10 Arbitrary reference trajectory

FBE convergence requires two things:

1. That $y_d(0) = y(0)$, i.e. that the reference trajectory starts at the actual system starting state. In general, any point where y is constrained to be different from y_d will be incorrectly trained, i.e. there will be divergence instead of convergence.

2. That the reference trajectory is continuous, i.e. that its derivative is bounded above and below. This is obviously necessary if the reference derivative must be computed. However, it is still a requirement in the modified FBE system where t is A's only input, because of the previous requirement. If y_d has a discontinuity then there will be a region after that where y will not be able to match the desired state.

Figures 4.8–4.10 show what happens when these requirements are violated.

Figure 4.8 shows the result of 1000 training iterations when the actual and reference starting states differ. The position oscillates around the reference, and the oscillations increase with further training.

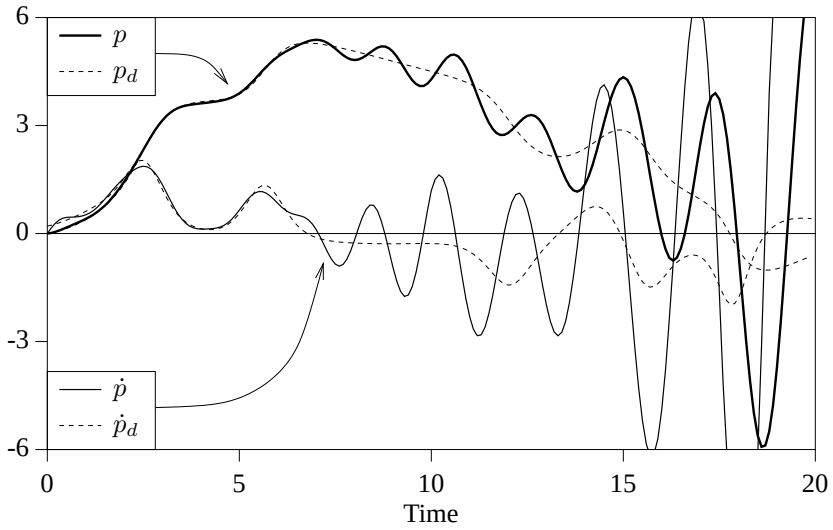

Figure 4.7: *Partial training of the system described in figure 4.6 (600 iterations). Convergence has not been achieved for later times.*

Figure 4.9 shows the result of 1000 training iterations with a discontinuous reference trajectory. The position has mostly converged to its reference but the velocity is oscillating about its reference, especially near the discontinuity. Figure 4.10 shows the same system after 3700 training iterations. The velocity oscillations have increased and are causing noticeable oscillations in position as well. This system continues to diverge with further training.

Why are the reference trajectory requirements necessary to prevent divergence? Consider the points where y is somehow forced to be different from y_d (these will be called "mismatch points"). This obviously happens when the starting reference and actual states differ. It also happens when there is a reference trajectory discontinuity, because no amount of control effort is sufficient to track such a step change. At the mismatch points the error e will always be nonzero. The error e integrates x_a, so x_a at these points will get larger each time the trajectory passes through them. Now, x_a acts to restore the actual state to the reference, but there is no "braking" force applied and so the reference is overshot. The training process does not provide any limiting influences on x_a, so the overshoot gets larger over time. The effects of the overshoot propagate out from the mismatch points, resulting in oscillations. This effect will be called the "over-training problem".

To clarify the problem: When the actual state is close to the reference trajectory, the error value x_q has the meaning "this is what x_a should have been". Elsewhere x_q is just acting as a restoring force. These two different meanings of the error signal (corrective and restorative) conflict.

4.11 Output limiting

One way to solve the over-training problem is to limit the magnitude of x_a somehow. This can be done by training x_a towards zero after each time step (this is called "output limiting"). Thus there will be two competing effects at the mismatch point: the error training from e and the target training towards zero. Eventually an equilibrium will be reached, at which point training will have converged.

This is demonstrated in figure 4.11, which shows the system of figure 4.6 with a reference discon-

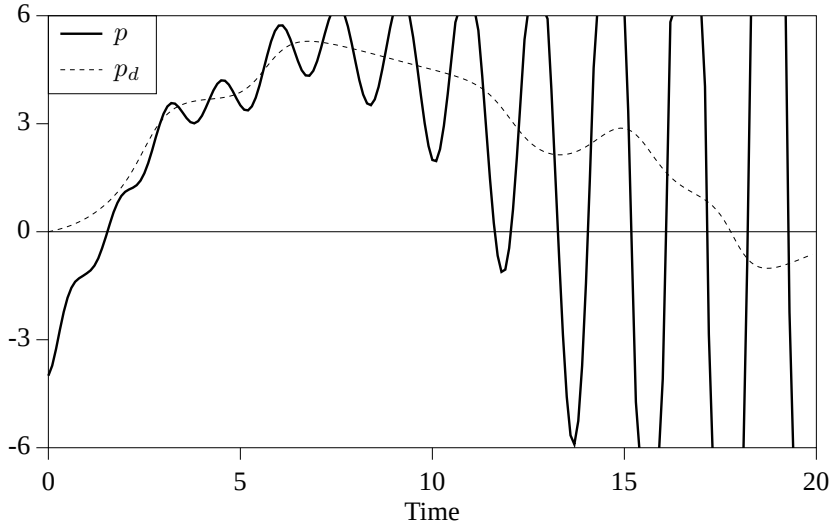

Figure 4.8: *The system of figure 4.6 when the actual and reference starting states differ (1000 iterations).*

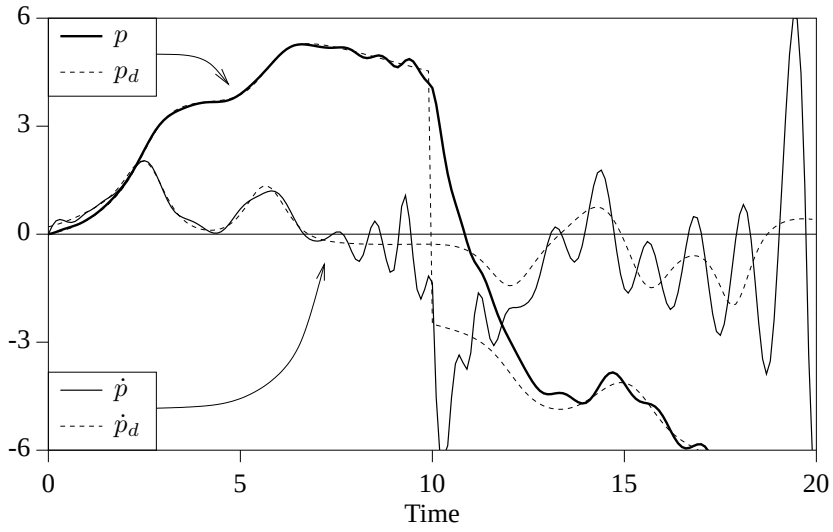

Figure 4.9: *The system of figure 4.6 when the reference trajectory has a discontinuity at* $t = 10$ *(1000 iterations).*

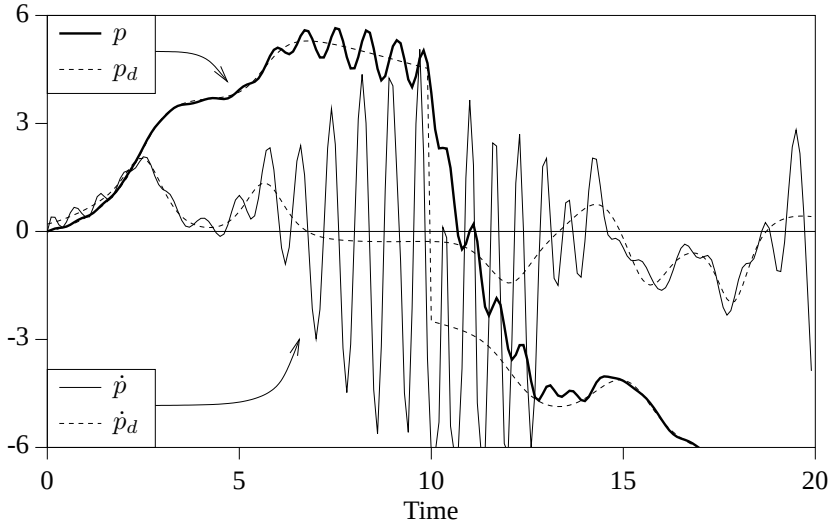

Figure 4.10: *The system of figure 4.6 when the reference trajectory has a discontinuity at* $t = 10$ *(3700 iterations).*

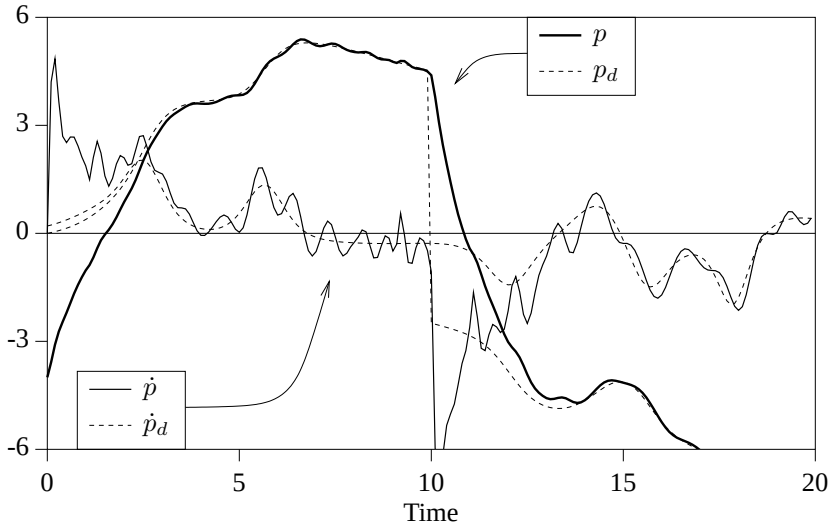

Figure 4.11: *The system of figure 4.6 with a reference discontinuity, an incorrect starting state, and output limiting with a learning rate of 0.002 (5000 iterations).*

tinuity and an incorrect starting state. Output limiting has been used with a learning rate of 0.002. The graph was generated after 5000 iterations, at which point convergence had been achieved. The position tracks its reference reasonably well, except around the mismatch points. The rate at which the position reacquires the reference after a reference discontinuity is dependent on the output limiting learning rate.

4.12 Conclusion

The feedback-error (FBE) control scheme has been described, and it has been shown that its utility is limited by the fact that it requires (1) a known starting state and (2) a continuous reference trajectory to be generated by an external agent. These problems can be partially solved by output limiting, although this is unsatisfactory.

FBE is a starting point for the development of the FOX controller: the next chapter will derive FOX from a first-principles analysis of the problem that FBE is trying to solve. It will be shown that the FOX algorithm can be regarded as a generalization of FBE. Experiments on FOX will subsequently show that FBE has some hidden problems: in particular, it is only capable of controlling systems that have a "zero-order eligibility model".