LEARNING CONTROL OF COMPLEX SKILLS

by

Lara Sidonie Crawford

Memorandum No. UCB/ERL M98/53

24 September 1998

COVER

LEARNING CONTROL OF COMPLEX SKILLS

by

Lara Sidonie Crawford

Memorandum No. UCB/ERL M98/53

24 September 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Learning Control of Complex Skills

by

Lara Sidonie Crawford

A.B. (Harvard University) 1991
M.S. (University of California at Berkeley) 1993

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Biophysics

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor S. Shankar Sastry, Chair
Professor Lawrence Stark
Professor Steven Lehman

Fall 1998

Abstract

Learning Control of Complex Skills

by

Lara Sidonie Crawford

Doctor of Philosophy in
Biophysics
University of California at Berkeley
Professor S. Shankar Sastry, Chair

This dissertation presents a hierarchical controller which can learn to perform complex motor skills. Humans routinely coordinate many degrees of freedom smoothly and effortlessly to achieve complex goals. Moreover, we are good at learning new patterns of coordination to produce new skills. Robots and artificial systems, on the other hand, typically have difficulty with the kinds of behaviors that come most naturally to us. Skills such as running, skiing, playing basketball, or diving involve complex nonlinear dynamics, many degrees of freedom, and behavioral goals that can be difficult to specify mathematically; goals such as "ski down the mountain without falling down" or "shoot a layup" must be translated from linguistic requirements into dynamic system constraints. The focus in this dissertation will be on the skill of platform diving, in which the diver's goal is to execute a certain dive and enter the water in a fully-extended, vertical position. Controlling a simulated diver is a difficult problem for standard control and planning algorithms; conservation of angular momentum gives the system dynamics a nonholonomic constraint with nonlinear drift.

In this dissertation, ideas from the fields of biological motor control and learning are combined with new learning algorithms in the design of a hierarchical controller which learns to dive. At the lower level of the control hierarchy, each degree of freedom in the diver's joints is assigned a controller based on biological pattern generators for fast, single-joint movements. These controllers contain neural networks, which are trained on data generated by simulation. The higher level of the control hierarchy incorporates ideas from human skill learning: to achieve a desired behavior pattern, a human learning a new skill uses information from instructors and from watching other performers to build a mental model of the task requirements, and then practices to refine the parameters of this behavioral model. In the high-level controller, each dive is represented as a sequence of multi-joint synergies. The controller learns initial estimates of the timing of these synergies from observational data and then refines these estimates through Q-learning with repeated simulations.

Professor S. Shankar Sastry
Dissertation Committee Chair

Contents

List of Figures

List of Tables

Acknowledgments

I would first like to thank my advisor, Shankar Sastry, for his continued support and encouragement. His energy, enthusiasm, and zest for exploring new ideas have been inspirational. I would also like to thank Larry Stark and Steve Lehman for being on my dissertation committee and for giving me advice and suggestions over the last few years.

Thanks to Jessica Hodgins for giving me the physical human model which was used in this work, and to George Pappas for letting me use his code as the basis of the graphical visualization program for the diver. I am grateful to Ron Parr for several very helpful discussions of reinforcement learning and of the problems with applying it to the diver system. Thanks also to Robin Benedetti for her advice and insights into diving.

Thanks to the ONR Graduate Fellowship, the Berkeley Graduate Fellowship, the P.E.O. Scholar Award, and ARO under grants DAAL 03-91-G0191, DAAH 04-91-1-0119, AASERT DAAH 04-94-G0211, DAAH 04-95-1-0588, DAAH 04-95-1-0264, and MURI DAAH 04-96-1-0341 for financial support.

So many friends, colleagues, and officemates provided comradeship, advice, and a willingness to listen that it is impossible to list them all. My graduate school experience would not have been the same without them.

I am very grateful to my family for their lifelong encouragement; they made it possible for me to grow up always believing that some day I would earn a Ph.D. Finally, many thanks to my wonderful husband Dan for his unwavering support through all the frustrations and successes.

Chapter 1

Introduction

One of the amazing successes of biological systems is their ability to learn to coordinate many degrees of freedom smoothly and efficiently to produce complex behaviors. In particular, how humans learn to dance, ride a bicycle, or execute a dive has been a subject of psychological research for many years. Robots and other artificial systems, however, have traditionally had great difficulty with acquiring or manifesting these kinds of complex motor skills, which often involve intrinsically complicated dynamics, the coordination of many degrees of freedom, and behavioral goals which may be difficult to specify mathematically. However, by combining recent advances in theories of biomotor control, better learning algorithms, and faster computers, artificial systems may now acquire some of these advanced skills.

One of the problems in controlling or learning a complex skill is that the goal is a general behavioral requirement, often expressed linguistically. Statements such as "don't fall off the bicycle" or "rotate through one and one-half somersaults while in the pike position" need to be expressed analytically for learning and control in artificial systems. To represent these goal statements mathematically, it is necessary to understand what the important features of the skill being learned are, in a physical as well as linguistic sense. Also, as these behavioral goals do not deal with specific joint requirements, the controls that will achieve a behavioral task description are often nonunique. The twin questions of *what is controlled* and *what is learned* in human motion and skill acquisition have been addressed by many researchers in many different ways over the last century, and some of their insights can be useful in designing artificial systems.

The psychology literature has distilled three stages of motor learning, which remain largely unchanged since first proposed by Fitts in 1962 [26]. First is the cognitive stage, in which the task or skill is turned over in the mind, analyzed, and perhaps put into words. This is the stage during which the input of a teacher can be effective. In the second associative, or fixation, stage, the performer practices the skill, improving and growing more consistent until errors are rare. In the final, or autonomous, stage, the performer continues practicing the movement until it becomes automatic, and requires very little conscious thought. During this stage, performance accuracy increases beyond the level at which errors can be detected, and speed increases in tasks for which it is important. The performance of the skill gradually becomes more resistant to stress and outside disturbances. Also in the autonomous stage, there often seems to be a corresponding shift in control to lower hierarchical levels; less high-level feedback is used, and behavioral subskills are combined together and programmed as a unit [26], [90].

Relatively little is known about the mechanism of any of the three learning stages, however; much of the recent work in motor control and learning has focused on control, to the detriment of learning (see [30] for a review). In the first, "learning by watching," stage of skill acquisition, what information does the student extract from watching an expert or from a teacher's instructions? What does the student refine in the later, "learning by doing," stages? These questions are, at their foundation, questions of representation: how are complex skills represented in the human motor system? The information extracted from teaching examples is then the information most relevant to the skill representation. One possibility is that the learner extracts the kinematics of the movement being performed, as in motion capture. There are several arguments against this suggestion. For one, the amount of information to store would be huge. In addition, a purely kinematic movement representation is unwieldy; it is hard to see how it would be easily broken down into sub-movements in the second stage of learning. Also, this representation leaves unanswered the question of how the correct torques to produce the required kinematics are produced. For example, in walking it is clear that everyone uses slightly different kinematics; you can recognize a friend from a distance by the way he or she walks. Another possibility is that the learner extracts some kind of scaled joint torque information from teaching examples. This is also unlikely; it has been repeatedly shown in walking, for example, that even though the kinematics are fairly similar for two different subjects or for the same subject from one stride to the next, the joint torques and electromyogram (EMG) patterns are often different, especially at the knee and hip [25], [77], [109], [108].

A more likely scenario is that the representation is in the form of a behavioral or synergetic model, in which the controls are restricted to a parametrized family, producing a stereotypical output behavior. This kind of model meshes well with a hierarchical control architecture; a controller at a given level of the hierarchy merely needs to output tuning parameters for the controllers at the next lower level. Such a control structure, while perhaps making the system less flexible, simplifies the control and learning problem tremendously by reducing the number of possible controls and allowing them to be represented in a compact way. Generalizing from known to new tasks also becomes easier, simply by combining known synergetic structures in new ways with new coupling parameters. For example, in running or walking, the motion of the center of mass has a similar form across subjects, speeds, and terrains. In running, a spring-mass model for the stance leg explains most of the observable behavior. In walking, an inverted pendulum model for the stance leg is a good description. Good discussions of locomotion biomechanics can be found in [25] and [64]. The variability of the joint moments can be explained by postulating a support moment synergy. The muscles act together to produce a required support moment; the variation in the joint moments themselves are required by the slight changes in joint angles. A second synergy may act to balance the body's mass, and a third to place the swing foot. (See [25], [109], [108].) Thus, in running and walking, the controls are organized to produce three synergetic subgoals, which can be combined to produce the overall spring-mass or inverted pendulum behavior. Different speeds or stride lengths, for example, can be achieved by adjusting parameters within the three synergies, but the overall form of the controls does not change. Raibert [80], [79] used these ideas of behavioral synergies to build extremely successful running and hopping robots.

Other skills have been less studied than locomotion, but the results there tell a similar story. Vereijken, *et al.* [102], showed that in an artificial slalom task, what the subjects learned seemed to be the relative phasing between the periodic movements of the legs and that of the slalom platform. This data points to the phase being a parameter in a behavioral model of the coupling between the periodic leg motions and the periodic motion of the platform. Similarly, in springboard and platform diving, divers say that much of learning dives consists of learning the timing required between known submovements such as the going into and coming out of a pike. A behavioral model of a dive would therefore consist of a string of timed subbehaviors which combine to produce a desired topology: two and one-half somersaults, or one and one-half somersaults plus a twist, for example.

The idea of parametrized, synergetic control strategies applies at every level down the motor control hierarchy. For example, in multi-joint placement movements such as reaching movements, the coupling between the joints is often done in a simple, straightforward manner involving few parameters; the relationship between the shoulder and elbow kinematics in a reaching movement is fairly fixed, with only a few parameters such as the relative scaling changing with different movement variants [48], [46], [47]. Another example, spinal pattern generators, have long been known to be involved in many animals for such rhythmic behaviors as running, chewing, swimming, and breathing [35], [36], and, more recently, similar control mechanisms have been postulated for fast, goal-directed, single-joint movements [34], [48], [103], [43], [44], [5]; see Sections 2.4–2.5. The role of pattern generators is to produce stereotypical movements or control signals which can be varied through tuning parameters signaled by higher levels in the control hierarchy. A behavioral control strategy at a given level of the control hierarchy can be used as a module or subprogram at a higher level of the hierarchy; thus, the balance synergy or the support synergy becomes part of the walking behavior, multi-joint leg and arm movements combine to become the action of going into the pike position in a dive, which in turn is part of the overall dive behavior, and pattern generators for single-joint movements are coupled together to produce the multi-joint control synergy.

With a behavioral model for complex skill acquisition, "learning by watching" may consist of determining the structure of the task in terms of a behavioral model and providing an initial set of parameter estimates. Then, during the second learning stage, or associative stage, the performer practices the skill to refine the parameter estimates. New ways to parcel the skill into lower level behaviors, synergies, or patterns (sometimes also called "movemes," in analogy to linguistical phonemes) may become clear during this stage, simplifying the high-level control by moving the burden to lower levels [90]. This kind of parceling may also involve the development of an internal model for external task dynamics [92]. During the third learning stage, or autonomous stage, the movement becomes more and more automatic. The combination of parceling control to lower levels and making the movements more automatic corresponds to a shift from closed-loop to open-loop control (see Section 2.5).

In this dissertation, I focus on the skill of platform diving. As with all behavioral goals, a student diver is not given a set of explicit desired trajectories to follow; rather, the dive is described as a set of desired twisting and somersaulting rotations together with a desired end configuration (enter the water in a fully-extended, vertical position) and possible intermediate configurations (for a "pike" dive, the diver must go into the pike position during the dive). To execute such a dive, the diver must put together a string of lower-level behaviors, or patterns of coordination, with the right timing to achieve the desired rotations and enter the water correctly. The lower-level behaviors are such multi-joint actions as entering the pike position or executing the "throw" with the arms that will convert somersaulting velocity into twisting velocity [27]. The diver learns what the string of behaviors is from an instructor (for example, first throw, then pike, then come out of the pike into the layout position); this information forms the basis of the behavioral model for the dive. The diver can also get initial estimates of the timing of the various behaviors by more teacher input or by watching other divers. The timing is then refined through repeated practice.

The diving problem has intrinsically interesting dynamics as well as many degrees of freedom to coordinate. After the diver has left the board, he or she is subject to angular momentum conservation, which creates a nonholonomic constraint in the system dynamics. The diver leaves the board with some initial (non-zero) angular momentum, so the system has drift. The drift velocity depends on the configuration of the diver. Since the diver is falling while executing the maneuvers, there is a predetermined length of time in which controls can act. These features, combined with the behavioral nature of the goal, make the diving problem a difficult one for traditional control methods, as will be discussed in Chapter 3.

Further background on motor control and learning from the biological and psychology literature is provided in Chapter 2. Chapter 3 presents an analysis of the diving problem and describes the implementation of the diver in simulation. The learning controller design and the algorithms it uses are presented in Chapter 4, along with some simulation results. Finally, Chapter 5 provides a discussion of the simulation results as well as various open questions regarding the control design and the learning algorithms.

Chapter 2

Biological Background

Biological motor control systems have evolved over the millenia to be amazingly effective, versatile, and adaptable. They are also, however, extremely complex, and are not understood fully even at the lowest level of the control hierarchy, the muscles. In spite of our lack of understanding, some principles have emerged in the study of biological systems that are useful for designing improved artificial controllers. Much of the material in Sections 2.1–2.3 is based on Brooks [21] and McMahon [64].

2.1 Motor Control Hierarchy

The motor control system in vertebrates is structured in a hierarchical manner. (See Figure 2.1.) The limbic system provides the motivational impetus to start a movement. This system, the seat of emotions and biological drives, is essential for understanding the requirements of the task at hand and thus for insightful learning. The sensorimotor system is typically divided into three hierarchical levels. At the highest level, the association cortex, which is responsible for recognizing and attending to events and objects, creates the general plan of the movement, and the motor cortex provides specific tactics and directional information for the execution of the movement. The motor cortex is organized in a somatotopic map. Each area of the map seems to specify very simple movement or muscle patterns in the corresponding part of the body [51]. In the middle level of control, the cerebellum performs coordination and fine tuning of different components of the motor plan based on reports from the somatosensory, vestibular, and visual systems. Some of the most basic types of motor learning occur in the cerebellum. Also at the middle level are

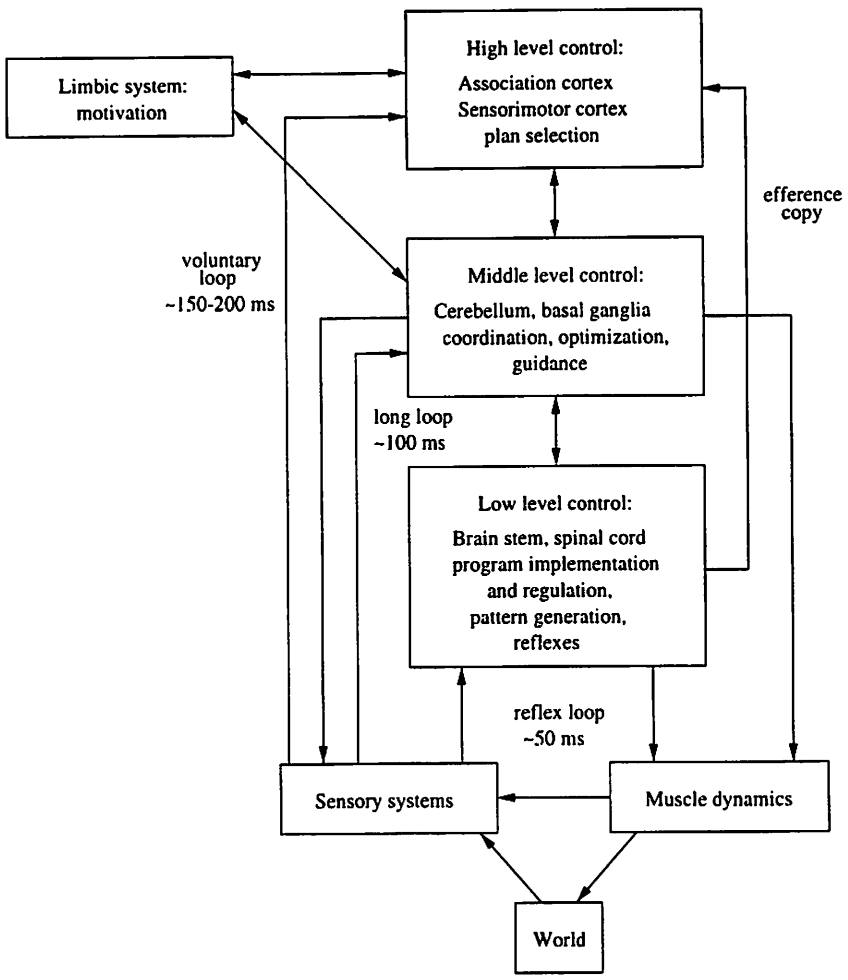


Figure 2.1: Hierarchical structure of the motor control system.

the basal ganglia, whose function is not fully understood; it is believed that they may act to scale motor plans and to group sequences of actions into automatic programs (as in the associative learning stage discussed in Chapter 1). They may also have a role in stabilizing the motor control system [51]. At the lowest level of the control hierarchy, the spinal cord implements the final motor command through the alpha motoneurons. Basic reflex loops are made through the spinal cord, and central pattern generators reside there as well. At the bottom of the hierarchy are muscles themselves (see Section 2.2) and the kinesthetic sensors (see Section 2.3).

At each level of the hierarchy, information is fed back to higher levels. Neurons that take information, usually control signals, down the hierarchy, are called efferents, while those that send signals like sensory feedback up the hierarchy are called afferents. In one particular type of feedback, one level of the hierarchy sends a copy of its commands to the lower levels back up to higher levels. Feedback of this kind is called efference copy or corollary discharge. Efference copy can be used as a control input to a model maintained at the higher level; such a model can then anticipate the outcome of the lower-level actions and thus alleviate the problem of slow feedback loops to some extent.

There are three main types of feedback loops indicated in Figure 2.1. The shortest of these are reflex loops, in which sensory information is transformed in the spinal cord into a reflexive motor command. From the sensory disturbance to the appearance of the motor response, these loops take on the order of 50 milliseconds. One of the most basic reflexes is the stretch reflex, in which stretching a muscle evokes reflexive contraction of that muscle. The stretch reflex is the one tested by doctors when they strike the patellar tendon with a mallet to stretch the quadriceps muscles. Reflexes will be discussed in more detail in Section 2.3. An intermediate feedback loop through the middle level of the hierarchy gives rise to motor output called a long-loop response or a functional stretch reflex. The output from this loop takes on the order of 100 milliseconds to become apparent. The long loop feedback path is used for motor set, the pre-setting of responses to postural disturbance. The response can be set, for example, to compensate for an anticipated disturbance, and thus can act functionally like the stretch reflex. The outermost feedback loop is the slow loop of voluntary responses. This loop provides feedback to the highest level of the hierarchy, the cerebral cortex, and takes about 150 to 200 milliseconds to produce a motor output.

A more detailed discussion of the motor control hierarchy can be found in Brooks [21].

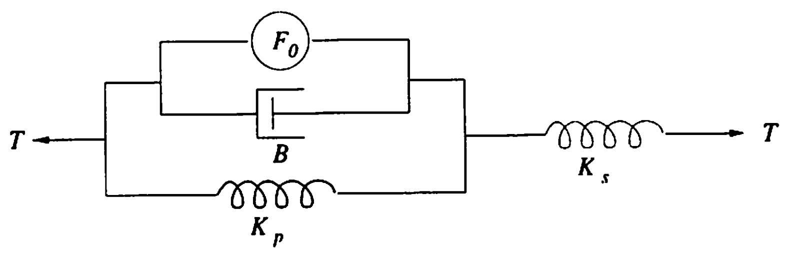


Figure 2.2: The Hill muscle model, which consists of a force generating element F_0, nonlinear parallel and series elastic elements K_p and K_s, and a nonlinear damper B, which together produce tension T.

2.2 Muscle Dynamics

Each muscle fiber in the body is innervated by an α-motoneuron. One motoneuron can innervate many muscle fibers spread throughout the muscle; a motoneuron and all its target fibers together are called a motor unit. When a pulse of activation travels down the motoneuron axon and reaches the synapse, the neurotransmitter acetylcholine is released into the synapse. The acetylcholine causes excitation of the muscle fiber membrane, which in turn evokes a burst of force production called a twitch. If the neuron transmits pulses at a high enough frequency, the twitches will overlap and begin to sum together. If the activation pulses arrive at a still higher frequency, the twitches will fuse together to produce a constant force. When this occurs, the muscle is said to be tetanized.

In 1938, A. V. Hill developed a lumped-element model of muscle based on several classic experiments in muscle dynamics [40] (see Figure 2.2). The model consists of a force generating element in parallel with a nonlinear damper and a nonlinear elastic component along with a second nonlinear elastic component in series with the other elements. The time history of the tension produced by the force generator (also called the active state) is not specified by the model; filtered square steps or pulses are frequently used in simulations, however. The Hill model, though it has several drawbacks, limitations, and inaccuracies, is still widely used to model muscle, since it captures most of the major characteristics of shortening muscle behavior and is easier to simulate than are other more mechanistic models like Huxley's 1957 model [50].

More detail about muscle physiology and dynamics can be found in [64]. In this work, however, all control actions will be performed at the level of joint torque or velocity.

2.3 Sensors and Reflexes

There are several types of proprioceptors used by biological organisms to sense the state of their muscles and joints. One of the most important is the spindle organs. The spindles are "intrafusal" muscle fibers scattered among the regular, extrafusal fibers in the muscle. The ends of the intrafusal fibers are attached to the muscle, so the spindle is in parallel with the muscle fibers. The spindle's output is carried by Ia and II neurons. The spindles sense the length and velocity of the muscle, and may also function as "event detectors" because the Ia ending is most sensitive to the initial change in muscle length [3]. Spindles only respond to muscle lengthening beyond a certain spindle rest length, which is set by γ-motoneuron inputs. This "fusimotor" system allows the spindles to maintain their sensitivity at different muscle lengths.

The stretch reflex, mentioned in Section 2.1, depends on negative feedback from the spindle organs, sometimes called stretch receptors. The Ia fiber, carrying the spindle signal, makes an excitatory synapse on the α-motoneuron innervating that muscle fiber. Thus, a stretch of the muscle excites the spindles, which in turn excite the α-motoneuron, which causes the muscle to contract. In order to avoid this reflex during voluntary movements, α- and γ-motoneurons are typically coactivated so as to keep the spindle set point at the current muscle length.

Another important proprioceptor is the Golgi tendon organ. These sensors are located in series with the muscle, close to the muscle-tendon junction, and sense force. Their output is carried by Ib fibers to interneurons (required to change the sign of the signal) which synapse on the α-motoneuron. This feedback loop, as well as the stretch reflex, is shown in Figure 2.3. The α- and γ-motoneurons and the spinal interneurons all receive control signals from higher centers.

The final proprioceptor type to be discussed here is the joint receptor. These receptors are located in joint ligaments and sense joint angle. Each receptor has a preferred angle to which it responds, so the joint angle is coded by the responses of a population of receptors.

An important feature of the lowest-level reflexive connections is reciprocal inhibition, in which the stretch of one muscle inhibits the contraction of opposing muscles. Since each type of neuron can make only excitatory or only inhibitory connections, interneurons are needed to effect this inhibition. As shown in Figure 2.4, spinal interneurons, the mediators of reciprocal inhibition, also receive descending control inputs. Reciprocal inhibition acts

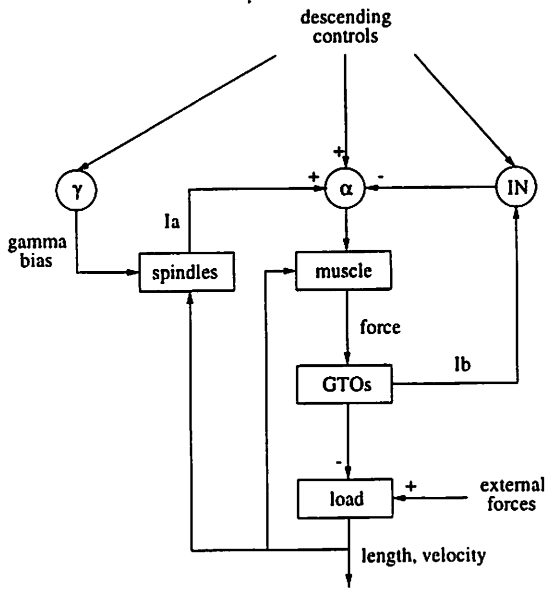


Figure 2.3: Block diagram of the stretch reflex and Golgi tendon organ feedback loops, modified from Houk's [49] much-reproduced diagram of the motor servo. IN = interneuron (note the change in signal sign); GTO = Golgi tendon organ.

as a negative feedback loop. When α-motoneuron activity causes the flexor to contract, for example, the extensor is stretched. This stretch is sensed by the spindles in the extensor, which excite the both the extensor α-motoneuron and the extensor interneuron through the Ia fibers. The extensor interneuron has an inhibitory connection to the flexor α-motoneuron and the flexor interneuron, which inhibits the extensor motoneuron. The effect of the extensor stretch is thus both to excite the extensor and to inhibit the flexor.

Also shown in Figure 2.4 are Renshaw cells, a type of interneuron which act to promote co-contraction (which normally occurs mainly in slow movements or small, accurate movements) through negative feedback to the inhibitory interneurons. When the flexor α-motoneuron is active, a collateral excitatory connection to the flexor Renshaw cell tends to activate it as well. The Renshaw cell has inhibitory connections to both the flexor α-motoneuron and the flexor interneuron; the former serves to decrease flexor activation, while the latter releases inhibition of the extensor. The combination of these two effects is increased co-contraction of the flexor and extensor. The extent to which reciprocal inhibition or co-contraction occur is modulated by the descending inputs.

Many other, more specialized reflexes exist to assist specified tasks. For example, the placing reaction, in which a touch to the top of the foot provokes leg flexion, helps avoid

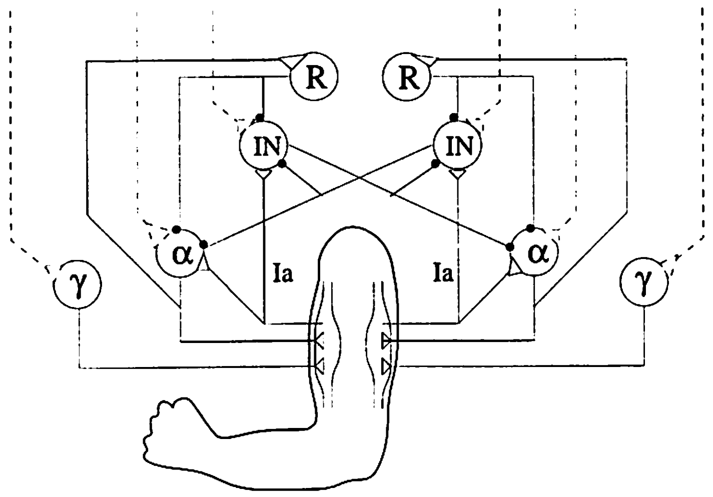


Figure 2.4: Reciprocal connections. The dotted lines represent descending inputs. Connections ending in triangles are excitatory, while those ending in dark circles are inhibitory. IN = interneuron; R = Renshaw cell. This figure is modified from Brooks [21].

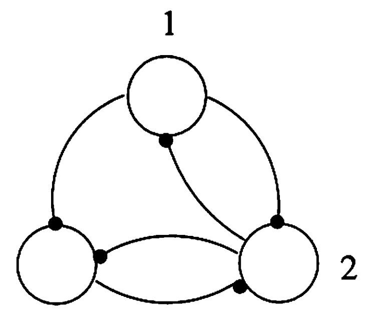
tripping while walking.

2.4 Pattern Generators

Also at the spinal level resides another type of control mechanism, the pattern generator. Pattern generators are groups of interconnected neurons that produce a stereotypical, often periodic, output. This output can be modified or switched on and off by descending controls from higher levels. As discussed in Chapter 1, pattern generators are an example of a behavioral control structure; they simplify control for higher levels in the motor hierarchy, which need only supply tuning parameters for the low-level pattern generators. Restricting the final control output to the family of controls producible by the pattern generator also solves the problem of choosing between nonunique controls for performing a task. The cost, of course, is that restriction to a control family allows less flexibility; the controls chosen may not be optimal for a specific task. Pattern generators have been identified in many animals for rhythmic movements like breathing, walking, swimming, and chewing. (Reviews can be found in [35] and [36].) Some typical pattern generators are shown in Figures 2.5 and 2.6.

More recently, several investigators, including Gottlieb, Corcos, and Agarwal (1989) [34], Hannaford and Stark (1985) [37], Wadman, *et al.* (1979) [103], and Hoffman and Strick

A.



B.

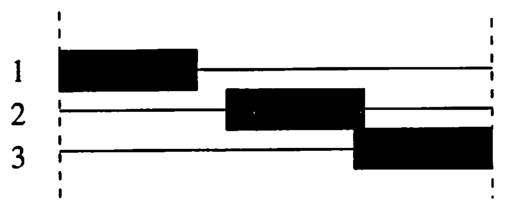


Figure 2.5: A. Simplified representation of the pattern generator in the lobster stomatogastric ganglion, a well-understood system which controls the rhythmic movements of the digestive tract, from [66]. Each circle represents a group of neurons of similar function. The neurons in group 1 produce rhythmic output on their own, with a frequency that is determined by descending controls, although such a pacemaker is not necessary for a group of neurons like this to produce sustainable oscillatory output. All the connections shown are inhibitory. B. Schematic of the phasing of the activity of these neurons in the lobster system, adapted from [66].

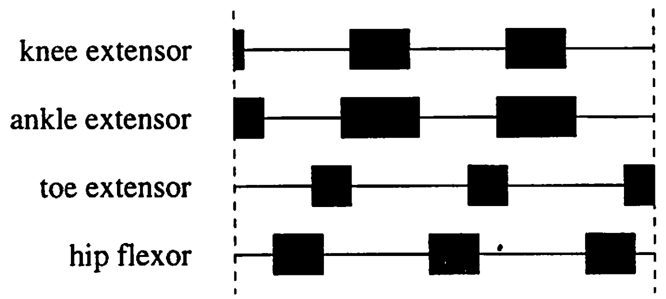


Figure 2.6: Bursting pattern of electromyogram recordings in the hindlimb of a decerebrate cat walking on a treadmill, adapted from [36]. A decerebrate animal is one whose nervous system has been transected at the level of the midbrain, and whose remaining motor functions are thus performed without the benefit of cerebral input. The output of the spinal pattern generator can be observed much more clearly in this type of animal.

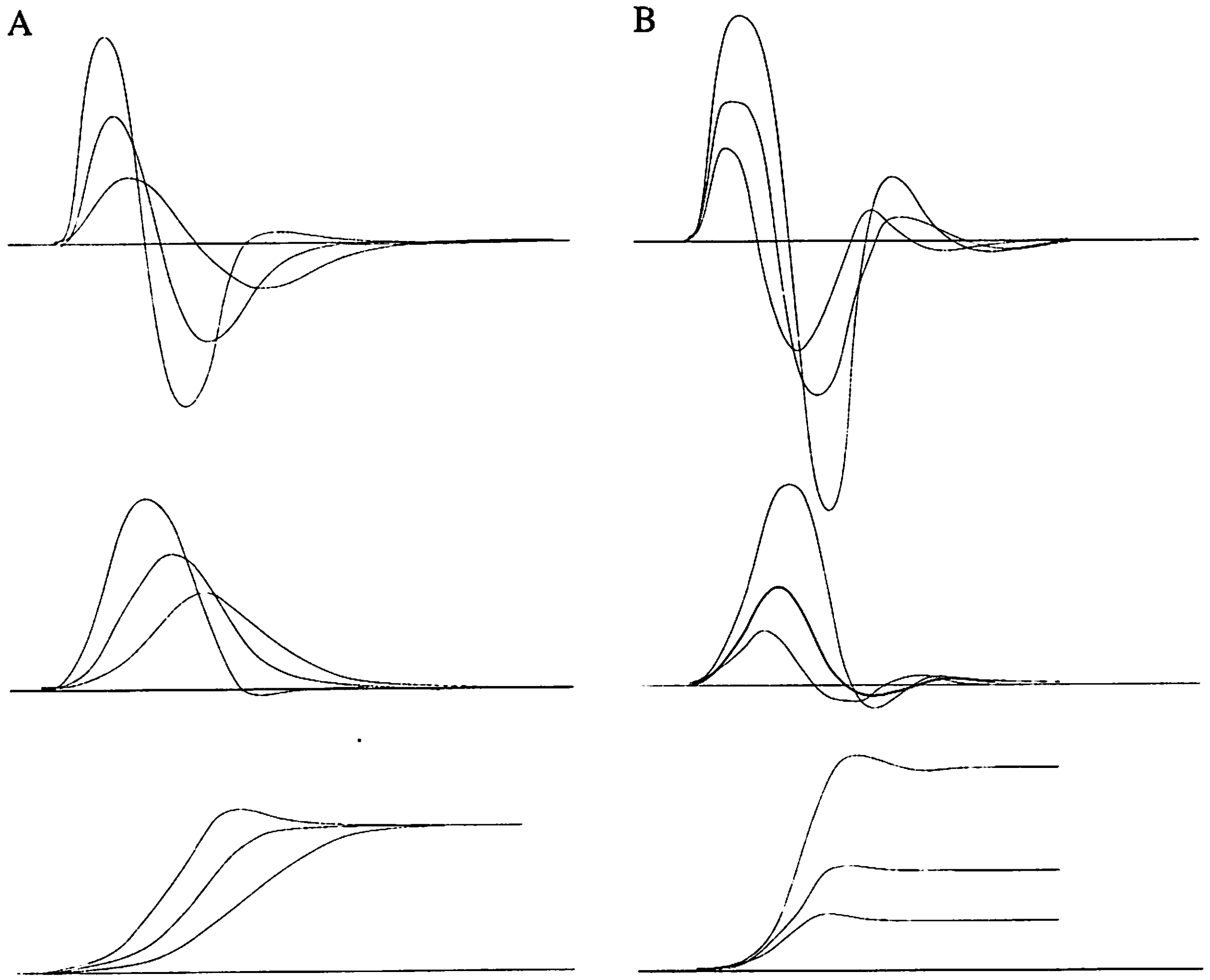


Figure 2.7: Sketch of fast, goal-directed single-joint movements. The top diagram in each column shows torques, the middle diagram shows velocities, and the bottom diagrams shows positions. All plots are sketches based on elbow flexion data presented in [34]. A. Movements at three different speeds. B. Movements of three different distances, 18°, 36°, and 72°.

(1986; 1990) [43] [44], have proposed similar low-level control mechanisms for fast, goal-directed, single-joint movements. This type of movement exhibits a stereotypical double- or triple-burst EMG trace (agonist, antagonist, agonist) as well as a bell-shaped velocity profile (see Section 2.5). Diagrams of examples of these movements are shown in Figure 2.7. In general, the first agonist EMG pulse initiates the movement, the antagonist pulse provides braking, and the second agonist pulse serves to clamp the movement [37]. The first pulse is feedforward in nature, while the last two may have some feedback component involving the proprioceptive systems discussed in Section 2.3, though the origin of these later two pulses is still being debated [82], [52]. In some fast, goal-directed movements, in fact, the third or even both the second and third EMG pulses may be absent; the reasons for this are still not totally clear [52], [81], [65]. One model that has been proposed to encapsulate the observations of many researchers and explain the control of these fast, single-joint movements is that of Gottlieb, Corcos, and Agarwal [34]. Their controller generates such movements by producing a pattern of rectangular activation pulses, which are then filtered through the motoneuron pool and the muscles to produce the joint torque and the associated EMG signal. The issue of whether the pulses are feedforward or feedback in origin is ignored. An appropriate movement would be produced by simply varying the heights and widths of these activation pulses. Gottlieb, Corcos, and Agarwal and their colleagues ([34], [33], [48]) have described situations in which the biological controller for single-joint movements seems to change the height of the first control pulse (varying the intended movement duration), and others in which it seems to change the width (varying the load). Their work has also extended to factors influencing the latency before the second, antagonist pulse (which seems to vary with both load and movement duration), but less is known about the antagonist pulse itself. Gottlieb [33] proposed a model for the agonist and antagonist pulse heights H_{ag} and H_{ant}, pulse width W (for both agonist and antagonist), and antagonist latency T_{ant} (see Figure 2.8). The antagonist latency is expressed in terms of the movement extent X, defined as $D + k_x J$ (where D is the target distance and J is the inertia of the limb plus load), and effort F, defined as $k_f U T_e$ (where U is the perceived movement urgency, a measurement between zero and one which is roughly related to desired movement speed,

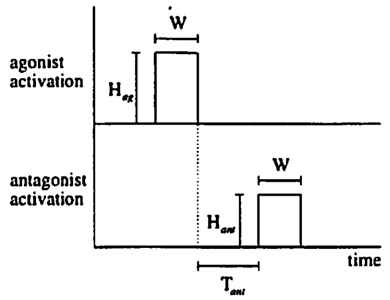


Figure 2.8: Diagram of the pulse parameters used in Gottlieb's pulse activation model [33].

and T_e is the target size).

$$\begin{aligned}
W &= W_0 + c_0 X \\
H_{ag} &= c_{ag} F (1 - e^{-\frac{X}{X_0}}) \\
H_{ant} &= c_{ant} F (1 - e^{-\frac{X}{X_0}}) e^{-\frac{D}{D_0}} \\
T_{ant} &= T_0 + c_T \frac{X}{F}
\end{aligned} \tag{2.1}$$

Note that the agonist and antagonist pulses share the same width. Gottlieb designed the structure of Equations (2.1) based on experimental observations like those described above, and indeed simulated movements using this model correlate relatively well with observed data. One interesting feature that Gottlieb points out is the change in control strategy over the range of X; for small X, the agonist pulse height is approximately proportional to $\frac{X}{X_0}$, while for large X, the pulse height is approximately constant with respect to X. A similar shift holds for the antagonist pulse height. Gottlieb interprets this strategy shift as arising from the fact that there is a physiological minimum to the pulse widths (given by W_0 in the model); for movements of very short duration, then, it is necessary to vary the pulse heights to achieve a wide range of movements. A control structure such as Gottlieb, Corcos, and Agarwal's pulse activation model can be seen as a pattern generator for fast, single-joint movements, as it produces a stereotypical output that can be modified by parameters indicating the pulse heights and widths, or, at a higher level, descending commands indicating the desired magnitude and speed of the movement. This is the type of pattern generator that will be incorporated into the hierarchical control design in Chapter 4.

2.5 Motor Control and Learning

Although a great deal of headway has been made in understanding low-level motor control and regulation, it is still unclear how the control works at higher levels. Bernstein, a pioneer in cybernetics and motor control (working in the 1930s–1960s), posed several basic questions that are still relevant to motor control today (see [107]), including the issues of how the motor control computations are organized in the brain, how these are translated into muscle activation signals, and how so many degrees of freedom as are present in a human are coordinated so smoothly. Though theories abound, much of the field, especially where learning is concerned, is still in the descriptive stage. The psychology literature describes several well-known "laws" of simple movements; many of these are summarized by Schmidt [90]. For example, Fitts' law describes the logarithmic relationship between the difficulty or accuracy of a movement and the time t required to perform it; for simple target-touching tasks, the law can be written as $t = k_1 + k_2 \log_2\left(\frac{2A}{W}\right)$, where A is the movement amplitude, W is the width of the target, and k_1 and k_2 are constants [90], [34]. There are also known exceptions to this rule. For example, in tasks requiring anticipation and timing, such as hitting a baseball, the inconsistency in the movement duration t increases as t increases and the movement velocity decreases; thus, in these tasks, faster movements are more accurate [90]. Other rules include force-variability principles (as the force required for a movement increases, so does the variability in that force) and impulse-variability principles (which relate the movement endpoint variability to the amplitude of the movement and the movement time). There are also features of movements that are typically observed; for example, for fast, single-jointed movements, the velocity profile of the joint is a stereotypical bell shape. For multi-joint movements, the point of greatest attention, such as the hand in reaching movements, generally has a similar velocity profile [21], [52].

There are features of motor learning that have been widely documented as well. For learning complex tasks, for example, there are the three stages discussed in Chapter 1: the cognitive stage, during which teaching is effective; the associative, or fixation stage, during which the learner gradually improves and becomes more consistent, and the autonomous phase, when the task becomes automatic and no longer requires full concentration [90], [26]. In infant motor learning, it has been observed that motor coordination is achieved at the proximal joints first, and then progresses to the more distal joints [73]. As mentioned in Chapter 1, there is also evidence that learning involves a shift from feedback toward feedforward control at multiple hierarchical levels. The outer feedback loop shifts downward in the control hierarchy, with the performer relying more on proprioceptive information and less on high-level sensory information such as vision [26]. Also, subskills are combined to form new, self-contained skills which are controlled as single units at the higher levels [90]. A decrease in the amount of feedback control over the course of learning has also been observed at lower levels for many different skills, including pistol aiming, non-dominant handwriting, dart throwing [73], slalom skiing [101], and walking in infants [99]. When beginning to learn these skills, the novice's joint movements are usually highly correlated or coupled with one another and quite stiff. As learning occurs, the joint motions become less correlated and more relaxed (see also [26]).

Given these observations, and knowledge about the underlying physiology, there have been several theories about motor control and learning that have been popular in the recent literature. There have been many reviews of the trends in motor control and learning research; a few can be found in [1], [72], [30], and [52] (Chapter 1). Though the approaches seem somewhat opposed along several dichotomies, including central *vs.* peripheral, open-loop *vs.* sensor-driven, programmed *vs.* emergent, and information processing *vs.* dynamics, all the approaches are valid under some conditions, and can be partly reconciled if the level of explanation is made clear and distinctions are made between regulation, control, coordination, and learning issues. In particular, a cognitive, programmed approach at the higher levels of the control hierarchy combined with a more dynamic explanation at the periphery may be able to draw many of the conflicting viewpoints into a coherent whole (see also [30]).

From the information processing point of view, one idea that has been appealing for a long time is the concept of a motor program. In this paradigm, an action is stored in memory as a program, perhaps with some variables to be filled in at the time of implementation. There have been differing opinions, however, on what variables are controlled in these programs. A popular idea is that motor programs contain the phasing or relative time intervals and relative force between different portions of a complex movement, so the movement can be scaled in size by scaling the force (with the basal ganglia); thus, handwriting on paper looks much the same as handwriting on a chalkboard, though completely different sets of muscles are used [90], [21]. The program is defined in terms of the most important object (for example, the writing implement), and complex movements may be broken into parts, or segments, and controlled by subprograms [21]. Similarly, new skills can be learned by putting together known movements [90]; the mechanism for this is largely unknown.

Schemas are a similar concept. Schmidt [89] originally defined a schema used to produce movement as a function describing the relationship between the control parameters and the movement outcome. The term "schema" has evolved in the literature somewhat, so that it may also mean a more complicated set of parametrized actions invoked by certain sensory stimuli or called by other schemas much like a subroutine (for example, see [4]).

In complex skills with clear goals, such as maximum height jumping, maximum speed pedaling, or posture regulation, it has been suggested that the motor control system activates its muscles in an optimal manner to achieve the goal (see [59] for a review). In walking, an optimality criterion based on the muscle forces may be used to determine which muscles are activated, and to what extent [77].

At lower levels, there is more data available, but there is still no agreement about specific mechanisms for coordination and control; for example, how are the controls chosen for a simple reaching movement, and why is the bell-shaped tangential velocity profile almost always evident in these movements? One explanation is that the motor control system has evolved to optimize a quantity such as total jerk in the movement, and the velocity profile of the hand in reaching movements or the joint in single-joint movements is a consequence of this optimization (see [5], [54]). To some extent, the joint torque profiles could even be simply scaled by the control system to produce different movements with similarly scaled velocity profiles at the hand [5]. Or, in order to coordinate the joints of the arm to produce a nearly straight trajectory at the hand, the control system may simply stagger and scale fixed velocity profiles for the different joints, though it is unclear under what conditions a bell-shaped tangential velocity profile would result at the hand [46], [47]. Another possibility is that planning is done in terms of force, and that the bell-shaped velocity profile is a byproduct of an underlying multi-joint torque-based coordination mechanism. One such mechanism would be one based on rectangular activation pulses, similar to the pattern generator for single-joint movements described in Section 2.4 [48]. In any case, the bell-shaped velocity profile may ultimately be due to a combination of feedforward control and low-level proprioceptive feedback mechanisms [37], [81], as mentioned in Section 2.4.

The prescriptive approach is not necessarily the best method of explanation, especially at progressively lower levels of the control hierarchy. At the lowest level, descending controls interact with the regulator composed of the fusimotor and Golgi tendon systems, as well as the muscular dynamics (see [21]), as discussed in Section 2.3 and mentioned above with respect to the triple-burst EMG profile. More recently, some researchers have taken a dynamical systems, bottom-up view, and have approached the problem in terms of the interaction of the motor control system with the environment. In this view, motor control involves setting the parameters of the muscle dynamics so that the desired movement is an equilibrium configuration of the organism-environment system. Learning in this scenario involves learning how the specified task dynamics relate to the environmental dynamics, and adjusting the motor system dynamics accordingly. For example, in a reaching task, if an external force field is imposed, the subject learns an internal model of the field and compensates for it [92]. The model appears to be in intrinsic coordinates rather in end-point coordinates; Shadmehr and Mussa-Ivaldi [92] postulate that the model is built up of a kind of motor control basis functions. These basis functions are simple force fields representing equilibria or other simple dynamic patterns; the end-point force is thus a function of the limb position and velocity. Learning to perform reaching movements in an external force field would entail adding compensatory basis functions to the normal reaching movement model.

In complex skill acquisition, learning can also be viewed as an adjustment in the internal movement dynamics to compensate for external dynamics. For example, in a task of learning to slalom on a specially-designed platform (mentioned in Chapter 1), the most important variable in achieving the task (maximum amplitude slaloming) is the relative phase between the subject's forcing of the springloaded platform and the motion of the platform itself. Learning this task consists of identifying the phase as the important variable, then adjusting it to maximize performance [102]. If the task does not involve new external dynamics, such as that of bimanual rhythmic finger tapping at the same frequency and a specified phase, the task specification itself may play the role of external dynamics, with the natural internal coupling relationships being adjusted to reflect the desired outcome [91]. The process of identifying the important task variables and couching the task goal in terms of the dynamics is as yet poorly explained by this emergent dynamics approach; such questions may be better addressed from a more cognitive point of view.

The approach to motor control taken in this dissertation is the idea of a hierarchical behavioral or synergetic control structure. As discussed in Chapter 1, a behavioral control structure is one which restricts controls to members of a parametrized family, thus producing stereotyped outputs but simplifying the control task for higher hierarchical levels. This approach incorporates and generalizes many of the control and learning ideas presented in this section as well as the previous section. A synergetic control structure is in fact a kind of generalization of the notion of a pattern generator (Section 2.4), which produces a stereotypical output which can be tuned by descending controls. The dynamical systems view that motor control consists of setting parameters in the muscle dynamics so that the desired movement is a system equilibrium can also be seen as an example of a behavioral structure; only particular parameters in the dynamics can be varied, so the higher level control is simply selecting among possible equilibria, thus specifying one of a family of possible control patterns. In particular, if the muscle dynamics can be specified with a set of motor control basis functions, as described above, setting the muscle dynamics simply means selecting the coefficients of the linear combination of bases. The control mechanisms for multi-joint reaching movements discussed above can also be viewed in a synergetic framework; whether the planning is done in terms of kinematics or force, the coordinating structure simply adjusts a few parameters in a fixed coordination scheme to produce a stereotypical reaching movement with a bell-shaped tangential velocity profile. The coordination scheme may be as simple as adjusting coupling parameters between lower-level control mechanisms, such as single-joint pattern generators, acting at each joint.

At a higher level, this coupling idea is similar to the dynamical systems approach to learning complex skills, as described above for the finger tapping and slaloming examples. In those examples, the learning mechanism simply learned the correct parameter, the coupling phase, for a control structure linking together lower-level controllers, in the case of finger tapping, or an internal control mechanism with environmental dynamics, in the case of slaloming. Motor programs and schemas for complex skills can also be viewed as behavioral or synergetic control structures. Both involve predefined classes of motor actions constructed for a specific task, but with some parameters allowing the task to be executed differently depending on variations in goal or external conditions. Also, motor programs and schemas fit easily into a hierarchical context, with higher level programs passing parameters to lower level subroutines, which can easily be recombined to make new skills. High level skills often involve several major subroutines, running either in parallel or in series. In walking or running, as discussed in Chapter 1, three subroutines run in parallel: the support, balance, and swing control synergies [109], [108]. For the support moment synergy, the muscles in the legs work together during the stance phase to keep the body from collapsing. The level of activity in different leg muscles may vary considerably from stride to stride, depending on small variations in the stride kinematics [25]. For the balance synergy, the muscles in the hips work to keep the torso upright. The swing control synergy is responsible for the swing and placement of the swing foot. These three tasks are somewhat independent, though they must be phased correctly with respect to each other and to external feedback such as ground contact. When combined together, these synergies produce the behavior of walking or running. At the highest descriptive level, walking and running themselves can be seen as behavioral control systems, with input parameters specifying such things as speed and stride length, and stereotypical output behaviors that are well described by an inverted pendulum for walking and a spring-mass model for running. The slaloming task can also be seen as the combination of two parallel subroutines, the forcing subroutine and the external platform dynamics subroutine; there, as discussed above, it is the phasing that is the critical coupling parameter. Platform diving is an example of a complex skill that involves subroutines running in series. For a one-and-one-half-somersault, full-twist dive, for example, the diver must first execute a "throwing" maneuver to tilt his or her axis of rotation, initiating the twisting movement, then enter the pike position, then exit the pike into the layout position. In diving, the critical coupling parameter between these behavioral subroutines is the timing, which is the most important variable in learning to execute a dive. The overall dive behavior can be viewed as a behavioral control system as well, with inputs specifying the number of twists and somersaults, for example, and stereotypical output movements that fall into a well-defined class of behaviors.

A hierarchical behavioral control structure, as defined here, encompasses much of what has been observed and proposed in the biological motor control and learning literature. Synergetic control structures act to restrict the number of ways of achieving a particular control goal, making the problem of controlling many degrees of freedom (Bernstein's most infamous problem) easier, and giving rise to invariant properties of the movement in the process (see [93]). Synergies, pattern generators, and motor control basis functions all serve as ways of simplifying the dynamics and restricting the options for the higher levels of control, and may be put together by the higher levels to achieve many different movements (see, for example, [29], [59], [23]). The way in which different synergetic control structures should be combined for a new skill can be learned during the cognitive stage; the synergetic parameters can then be fine tuned during the later two stages of learning.

Chapter 3

The Diving Problem

The skill of platform diving is an interesting one with which to test biologically-inspired control strategies. The dynamics of the diver system are inherently complicated due to a nonholonomic constraint with nonlinear drift, because it requires the coordination of a potentially large number of degrees of freedom, because the goal of the system is to achieve a desired pattern of behavior rather than a specified trajectory or goal configuration, and because the controls have a limited time (determined by gravity and the height of the board) in which to act. The control goal for the diver problem is: given a fixed set of initial conditions, execute a certain diving maneuver, such as a jackknife, a forward one-and-one-half-somersault pike, or a forward one-and-one-half-somersault, full-twist dive, and then enter the water in a fully-extended, vertical position. In general, the diving maneuvers are specified by the manner of takeoff from the platform, the number of somersaults and twists of rotation, and sometimes internal configuration information. With the problem specification used here, the platform takeoff is predetermined by the initial conditions. In general, however, the platform takeoff can be facing away from the platform with forward somersaulting rotation (forward), facing toward the platform with backward somersaulting rotation (backward), facing away from the platform with backward somersaulting rotation (reverse), or facing toward the platform with forward somersaulting rotation (inward). More complicated variants involving leaving the platform from a handstand position are also possible. The diver's initial angular velocity is entirely about the somersault axis for somersaulting dives; it can have a small component about the twist axis for twisting dives, but most of the twisting velocity is achieved by executing a "throwing" maneuver with the arms. The internal configuration information can specify that the diver must hold the pike

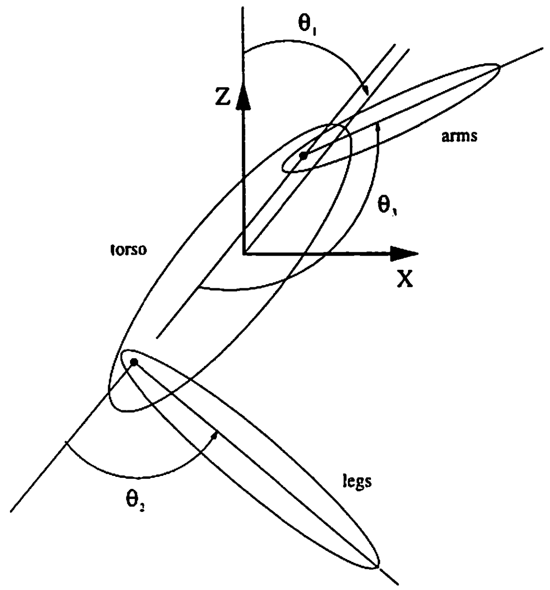


Figure 3.1: 2-D diver model, showing the shape space variables and overall orientation variables. The central mass models the body, the upper one the arms and the lower one the legs of the diver.

(bending at the hips and waist, with arms either out ("open pike") or holding onto the legs ("closed pike")) or tuck ("cannonball") position during the dive rotations. Thus, a forward one-and-one-half-somersault pike dive consists of leaving the platform facing forward, executing one and one-half somersaults (3π) of rotation in the forward direction while in the pike position, and entering the water in a fully-extended (layout), vertical position. In the control problem formulated here, the control task begins at the moment the diver leaves the platform; the takeoff is incorporated into the fixed initial conditions. Not allowing the diver to alter the initial conditions, the results of the takeoff from the platform, is an unrealistic simplification, as the takeoff is actually one of the most important things real divers have to learn.

3.1 Planar Diver Analysis

For an initial analysis, I have used a planar model of the diver. This two-dimensional model consists of three linked rigid bodies, for a total of five degrees of freedom (Figure 3.1). The planar model is sufficient for two-dimensional piking dives, such as the jackknife or the forward one-and-one-half-somersault pike.

The x and z directions do not affect the control of the system except to determine when the diver hits the water, so we can ignore them in the analysis. θ_2 and θ_3 are the shape space variables of the diver system; they describe the internal structure of the rotating body. θ_1 is a position variable of the system describing the overall orientation of the diver. The system has symmetry group S^1; that is, the Lagrangian is invariant under changes in θ_1. By Noether's theorem, then, we have a conserved quantity, namely the angular momentum of the diver:

$$\frac{\partial L}{\partial \theta_1} = 0 = \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_1} \tag{3.1}$$

and $\frac{\partial L}{\partial \dot{\theta}_1} = \mu$, a constant. Using the Lagrangian

$$\begin{aligned} L \;=\; & (\frac{\alpha_1}{2} + \beta\cos\theta_2 + \gamma\sin\theta_2 + \delta\cos\theta_3 + \epsilon\sin\theta_3 \\ & \quad + \zeta\cos(\theta_3-\theta_2) + \eta\sin(\theta_3-\theta_2))\dot{\theta}_1^2 \\ & + (\alpha_2 + \beta\cos\theta_2 + \gamma\sin\theta_2 + \zeta\cos(\theta_3-\theta_2) + \eta\sin(\theta_3-\theta_2))\dot{\theta}_1\dot{\theta}_2 \\ & + (\alpha_3 + \delta\cos\theta_3 + \epsilon\sin\theta_3 + \zeta\cos(\theta_3-\theta_2) + \eta\sin(\theta_3-\theta_2))\dot{\theta}_1\dot{\theta}_3 \\ & + (\zeta\cos(\theta_3-\theta_2) + \eta\sin(\theta_3-\theta_2))\dot{\theta}_2\dot{\theta}_3 + \frac{\alpha_2}{2}\dot{\theta}_2^2 + \frac{\alpha_3}{2}\dot{\theta}_3^2, \end{aligned} \tag{3.2}$$

we find that μ is of the form:

$$\begin{aligned} \mu \;=\; & [\alpha_1 + 2\beta\cos\theta_2 + 2\gamma\sin\theta_2 + 2\delta\cos\theta_3 + 2\epsilon\sin\theta_3 \\ & \quad + 2\zeta\cos(\theta_3-\theta_2) + 2\eta\sin(\theta_3-\theta_2)]\dot{\theta}_1 \\ & + [\alpha_2 + \beta\cos\theta_2 + \gamma\sin\theta_2 + \zeta\cos(\theta_3-\theta_2) + \eta\sin(\theta_3-\theta_2)]\dot{\theta}_2 \\ & + [\alpha_3 + \delta\cos\theta_3 + \epsilon\sin\theta_3 + \zeta\cos(\theta_3-\theta_2) + \eta\sin(\theta_3-\theta_2)]\dot{\theta}_3 \end{aligned} \tag{3.3}$$

$$=: \begin{bmatrix} b_1(\theta_2,\theta_3) & b_2(\theta_2,\theta_3) & b_3(\theta_2,\theta_3) \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} =: \mathbf{b}(\theta_2,\theta_3)\dot{\theta} \tag{3.4}$$

where α_1, α_2, α_3, β, γ, δ, ϵ, ζ, and η are constants; their values in the model used are given in Table 3.1. Conservation of μ, the angular momentum, is the single constraint for this system. This constraint is not integrable; that is, Equation (3.4) is not equivalent to any set of algebraic constraints on $(\theta_1, \theta_2, \theta_3)$. Nonintegrable constraints of this type are said to be nonholonomic.

In order to express the diver kinematics as a control problem, we can dualize the constraint (3.4) by considering the two-dimensional space of vectors which annihilate $\mathbf{b}(\theta_2, \theta_3)$.

Parameter	Value
m_t	51.779
m_l	27.694
m_a	9.124
$M = m_t + m_l + m_a$	88.597
I_t	3.290
I_l	1.992
I_a	0.323
j_{lx}	0.004
j_{lz}	-0.375
j_{ax}	0.047
j_{az}	0.190
l_x	0.043
l_z	-0.299
a_x	-0.012
a_z	-0.244
$\alpha_1 = I_t + I_l + I_a + \frac{m_l(M-m_l)}{M}(l_x^2 + l_z^2 + j_{lx}^2 + j_{lz}^2)$	
$+\frac{m_a(M-m_a)}{M}(a_x^2 + a_z^2 + j_{ax}^2 + j_{az}^2) - \frac{2m_lm_a}{M}(j_{lx}j_{ax} + j_{lz}j_{az})$	11.234
$\alpha_2 = I_l + \frac{m_l(M-m_l)}{M}(l_x^2 + l_z^2)$	3.732
$\alpha_3 = I_a + \frac{m_a(M-m_a)}{M}(a_x^2 + a_z^2)$	0.812
$\beta = \frac{m_am_l}{M}(-j_{az}l_z - j_{ax}l_x) + \frac{m_l^2}{M}(-j_{lx}l_x - j_{lz}l_z) + m_l(j_{lx}l_x + j_{lz}l_z)$	2.299
$\gamma = \frac{m_am_l}{M}(j_{ax}l_z - j_{ax}l_x) + \frac{m_l^2}{M}(l_{lx}l_z - j_{lz}l_x) - m_l(j_{lx}l_z - j_{lz}l_x)$	-0.343
$\delta = \frac{m_am_l}{M}(-j_{lz}a_z - j_{lx}a_x) + \frac{m_a^2}{M}(-j_{ax}a_x - j_{az}a_z) + m_a(j_{ax}a_x + j_{az}a_z)$	-0.645
$\epsilon = \frac{m_am_l}{M}(j_{lx}a_z - j_{lz}a_x) + \frac{m_a^2}{M}(j_{ax}a_z - j_{az}a_x) - m_a(j_{ax}a_z - j_{az}a_x)$	0.058
$\zeta = \frac{m_am_l}{M}(-a_xl_x - a_zl_z)$	-0.207
$\eta = \frac{m_am_l}{M}(a_zl_x - a_xl_z)$	-0.040

Table 3.1: Values of the parameters for the planar diver model, to three decimal places. m_t, m_l, and m_a are the masses of the torso segment, legs, and arms, respectively, in kilograms; I_t, I_l, and I_a are the corresponding inertias, in kilogram meters squared. (j_{lx}, j_{lz}) and (j_{ax}, j_{az}) are the coordinates of the leg and arm joints, respectively, in the zero position (origin at the center of mass of the torso segment), in meters. (l_x, l_z) and (a_x, a_z) are the coordinates of the leg and arm centers of mass with respect to their respective joints, again in the zero orientation, in meters. These values were calculated from the three-dimensional human model parameters given in Table 3.2.

This space represents the directions in which the system could move instantaneously if μ were zero. Since the diver's joint angles θ_2 and θ_3 can be controlled directly, a convenient choice of basis vectors for the null space of the constraint is

$$g_1(\theta_2,\theta_3) =: \begin{bmatrix} -\frac{b_2}{b_1} \\ 1 \\ 0 \end{bmatrix}, \quad g_2(\theta_2,\theta_3) =: \begin{bmatrix} -\frac{b_3}{b_1} \\ 0 \\ 1 \end{bmatrix} \tag{3.5}$$

Note that b_1 is always positive (see Table 3.1 and Equations 3.3–3.4). The system with arbitrary μ can now be written with two joint controls u_1 and u_2:

$$\begin{aligned} \begin{bmatrix} \dot\theta_1 \\ \dot\theta_2 \\ \dot\theta_3 \end{bmatrix} &= \begin{bmatrix} -\frac{b_2}{b_1} \\ 1 \\ 0 \end{bmatrix} u_1 + \begin{bmatrix} -\frac{b_3}{b_1} \\ 0 \\ 1 \end{bmatrix} u_2 + \begin{bmatrix} \frac{\mu}{b_1} \\ 0 \\ 0 \end{bmatrix} \\ &=: g_1(\theta_2,\theta_3)u_1 + g_2(\theta_2,\theta_3)u_2 + f(\theta_2,\theta_3) \end{aligned} \tag{3.6}$$

The final term, $f(\theta_2,\theta_3)$ is called the drift of the system; even when $u_1 \equiv u_2 \equiv 0$, the system (in particular, θ_1) continues to evolve. Note that the drift is a nonlinear function of θ_2 and θ_3. Since the Lie bracket $[g_1, g_2]$ is given by

$$[g_1,g_2] = \begin{bmatrix} \frac{1}{b_1^2}\left(b_3\frac{\partial b_1}{\partial\theta_2} - b_1\frac{\partial b_3}{\partial\theta_2} - b_2\frac{\partial b_1}{\partial\theta_3} + b_1\frac{\partial b_2}{\partial\theta_3}\right) \\ 0 \\ 0 \end{bmatrix}, \tag{3.7}$$

it is clear that $\{g_1, g_2, [g_1, g_2]\}$ spans the space (except at those isolated values of (θ_2, θ_3) where the first entry in $[g_1, g_2]$ is zero). The system is therefore locally controllable [76] even without making use of terms of the form $[f, g_1]$, $[f, g_2]$, etc., since the Lie bracket (3.7) is the same as the drift direction.

Returning to the dual constraint viewpoint, one can look at the diving problem as a Pfaffian exterior differential system defined by the ideal generated by the codistribution I:

$$I = \{\alpha\}, \tag{3.8}$$

$$\alpha = b_1 d\theta_1 + b_2 d\theta_2 + b_3 d\theta_3 - \mu dt \tag{3.9}$$

If we can convert this system into a standard chained form, sinusoidal or other steering methods that have been used for other systems can be applied (see Section 3.2). We have:

$$\begin{aligned}
d\alpha &= db_1 \wedge d\theta_1 + db_2 \wedge d\theta_2 + db_3 \wedge d\theta_3 \\
&= \left(\frac{\partial b_2}{\partial \theta_1} - \frac{\partial b_1}{\partial \theta_2}\right) d\theta_1 \wedge d\theta_2 + \left(\frac{\partial b_3}{\partial \theta_1} - \frac{\partial b_1}{\partial \theta_3}\right) d\theta_1 \wedge d\theta_3 \\
&\quad + \left(\frac{\partial b_3}{\partial \theta_2} - \frac{\partial b_2}{\partial \theta_3}\right) d\theta_2 \wedge d\theta_3 \\
&=: h_3(\theta_2,\theta_3)\, d\theta_1 \wedge d\theta_2 + h_2(\theta_2,\theta_3)\, d\theta_1 \wedge d\theta_3 + h_1(\theta_2,\theta_3)\, d\theta_2 \wedge d\theta_3 \\
&= -2\left(\gamma\cos\theta_2 - \beta\sin\theta_2 - \eta\cos(\theta_3-\theta_2) + \zeta\sin(\theta_3-\theta_2)\right) d\theta_1 \wedge d\theta_2 \\
&\quad -2\left(\epsilon\cos\theta_3 - \delta\sin\theta_3 + \eta\cos(\theta_3-\theta_2) - \zeta\sin(\theta_3-\theta_2)\right) d\theta_1 \wedge d\theta_3 \\
&\quad +2\left(-\eta\cos(\theta_3-\theta_2) + \zeta\sin(\theta_3-\theta_2)\right) d\theta_2 \wedge d\theta_3
\end{aligned} \tag{3.10}$$

$$\begin{aligned}
d\alpha \wedge \alpha &= (h_3 b_3 - h_2 b_2 + h_1 b_1)\, d\theta_1 \wedge d\theta_2 \wedge d\theta_3 \\
&\quad -h_3\mu\, d\theta_1 \wedge d\theta_2 \wedge dt - h_2\mu\, d\theta_1 \wedge d\theta_3 \wedge dt - h_1\mu\, d\theta_2 \wedge d\theta_3 \wedge dt \\
&= [2b_1\left(-\eta\cos(\theta_3-\theta_2) + \zeta\sin(\theta_3-\theta_2)\right) \\
&\qquad +2b_2\left(\epsilon\cos\theta_3 - \delta\sin\theta_3 + \eta\cos(\theta_3-\theta_2) - \zeta\sin(\theta_3-\theta_2)\right) \\
&\qquad -2b_3(\gamma\cos\theta_2 - \beta\sin\theta_2 - \eta\cos(\theta_3-\theta_2) \\
&\qquad\quad +\zeta\sin(\theta_3-\theta_2))] d\theta_1 \wedge d\theta_2 \wedge d\theta_3 \\
&\quad +2\mu(\gamma\cos\theta_2 - \beta\sin\theta_2 - \eta\cos(\theta_3-\theta_2) \\
&\qquad +\zeta\sin(\theta_3-\theta_2)) d\theta_1 \wedge d\theta_2 \wedge dt \\
&\quad +2\mu(\epsilon\cos\theta_3 - \delta\sin\vartheta_3 + \eta\cos(\theta_3-\theta_2) \\
&\qquad -\zeta\sin(\theta_3-\theta_2)) d\theta_1 \wedge d\theta_3 \wedge dt \\
&\quad -2\mu\left(-\eta\cos(\theta_3-\theta_2) + \zeta\sin(\theta_3-\theta_2)\right) d\theta_2 \wedge d\theta_3 \wedge dt
\end{aligned} \tag{3.11}$$

$$(d\alpha)^2 \wedge \alpha = 0 \tag{3.12}$$

So, α has Pfaffian rank 1 (except at isolated points where $d\alpha \wedge \alpha = 0$). In other words, the derived flag for this system looks like

$$\begin{aligned}
I^{(0)} &= \{\alpha\} \\
I^{(1)} &= \{0\}
\end{aligned} \tag{3.13}$$

as expected.

By Pfaff's theorem (see [76]), we can find local coordinates (z_1, z_2, z_3) so that

$$\{\alpha\} = \{dz_3 - z_2 dz_1\}. \tag{3.14}$$

To do this, we would need to find two functions q_1 and q_2 which satisfy

$$\begin{aligned} d\alpha \wedge \alpha \wedge dq_1 &= 0 \\ \alpha \wedge dq_1 &\neq 0 \\ \alpha \wedge dq_1 \wedge dq_2 &= 0 \\ dq_1 \wedge dq_2 &\neq 0 \end{aligned} \tag{3.15}$$

Then α can be written as:

$$\{\alpha\} = \{dq_2 - s dq_1\} \tag{3.16}$$

Using this coordinate system, the control system becomes:

$$\begin{aligned} \frac{dq_1}{dt} &= u_1 \\ \frac{ds}{dt} &= u_2 \\ \frac{dq_2}{dt} &= s\frac{dq_1}{dt} = su_1 \end{aligned} \tag{3.17}$$

The solutions to Equations (3.15) are not unique. It is notable that for the diver system $q_1 = t$ is not a solution, so the system is not feedback linearizable (see [76]). Simple time-scalings such as $q_1 = t + h(\theta_2)$ or $q_1 = t + h(\theta_1)$ also fail, as do $q_1 = \theta_1$, $q_1 = \theta_2$, and $q_1 = \theta_3$. The implications of this failure will be discussed in the next section.

3.2 Control Approaches for the Diver Problem

Although much work has been done recently on the control and steering of nonholonomic systems, most of it has been for drift-free systems (those for which $\mathbf{u} \equiv \mathbf{0}$ is an equilibrium point; for a survey, see [58]). Some specific cases with drift have been addressed, for example, left-invariant systems on SO(3) and GL(n) ([17], [6], [104], [85]), but very little work exists concerning general systems with drift. In the real world, systems with drift are common. For example, bodies in free-fall with some initial angular momentum have drift. The diver is an instance of this kind of system. After the diver has left the board, his angular momentum is conserved, but is generally not zero, leading to the drift term derived above. While in the air, the diver can change the drift velocity by changing his moment of inertia. He can also convert some of his somersaulting motion into twisting about the body's long axis by performing a "throwing" maneuver with his arms which shifts the body's angular velocity vector so that it is no longer aligned with the angular momentum. (See Frolich [27] for an analysis of the physics of various diving and trampoline maneuvers, and [68] for an analysis of the falling cat, a similar, though drift-free, problem in reorientation.)

3.2.1 Canonical Forms

Conventional techniques for controlling nonholonomic systems proved unsatisfactory for the planar diver model. For example, the diver model is an asymmetric version of the planar skater discussed in [105], which (when drift-free) can reorient itself arbitrarily by moving its two arms sinusoidally and out of phase with each other, using the chained form and the Pfaffian exterior differential system approach. Using the control system in Equations (3.17) and assuming initial coordinate values of zero, the controls

$$\begin{aligned} u_1 &= A\sin\omega t \\ u_2 &= B\cos\omega t \end{aligned} \tag{3.18}$$

yield the behavior:

$$\begin{aligned} q_1 &= -A\cos\omega t \\ s &= B\sin\omega t \\ q_2 &= \int_0^t s u_1 dt = AB\int_0^t \sin^2\omega t = \frac{AB}{2}\left(t - \frac{\sin 2\omega t}{2\omega}\right) \end{aligned} \tag{3.19}$$

Thus, at the end of each control period T, the directly controlled coordinates q_1 and s are back at zero and the steered variable q_2 has increased by $\frac{ABT}{2}$. This method of steering with sinusoids has been used for other nonholonomic systems such as cars with trailers and firetrucks as well ([71], [22], [100]), but these systems were all drift-free. The nonlinear drift in the diver system complicates the exterior differential systems approach considerably. As discussed in Section 3.1, for the diver problem it is not easy to find any solutions to Equations (3.15), although one is guaranteed to exist by Pfaff's theorem. None of the simple transformations that are commonly used are solutions to Equations (3.15), and variants on these choices also failed to produce solutions. Any controls generated by this method of course also cannot be guaranteed to produce a natural-looking (or even humanly possible) movement.

Another canonical representation for nonholonomic systems is the standard triangular form discussed by Kawski [56], [57] for nilpotent systems (systems with finite-dimensional Lie algebras in which all Lie brackets of order higher than a certain integer are zero). The representation can be extended to systems which can be made nilpotent through appropriate feedback; this class of nilpotentizable systems contains the class of systems that can be put in chained form. Nilpotent or nilpotentizable systems can be made finitely discretizable using a coordinate transformation, and are thus controllable using piecewise constant controls [28]. A general approach to controlling nonholonomic systems with drift that arise from Lagrangian systems with cyclic coordinates is explored in [31]. The approach discussed there also uses piecewise constant controls and takes advantage of the subtriangular structure of the system.

3.2.2 Optimal Control

An attempt was also made to steer the diver using the optimal control techniques developed by Sastry and Montgomery in [87], which minimize $\frac{1}{2}\int_0^T |u(t)|^2 dt$, but the resulting equations were extremely complex and would have required numerical solution. The method is as follows. In [87], Sastry and Montgomery derive coupled differential equations for the optimal controls of a system of the form

$$\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x) u_i \tag{3.20}$$

using the Hamiltonian

$$H(x,u,p) = p^T \left(\sum_{i=1}^{m} g_i(x) u_i \right) + \frac{1}{2} \sum_{i=1}^{m} |u_i|^2 \tag{3.21}$$

where x is the vector of generalized coordinates and p is the vector of associated momenta. For the $m = 2$ case, their result looks like:

$$\begin{aligned}
\begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \end{bmatrix} &= \begin{bmatrix} 0 & p^T[g_1, g_2] \\ p^T[g_2, g_1] & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} -p^T[f, g_1] \\ -p^T[f, g_2] \end{bmatrix} \\
u_i &= -p^T g_i(x) \\
\dot{p} &= -\frac{\partial f}{\partial x}^T p + \frac{\partial g_1}{\partial x}^T p(p^T g_1(x)) + \frac{\partial g_2}{\partial x}^T p(p^T g_2(x)) \\
\dot{x} &= f(x) - g_1(x)(p^T g_1(x)) - g_2(x)(p^T g_2(x))
\end{aligned} \tag{3.22}$$

For the diving system, from Equations 3.22 we get immediately that $\dot{p}_1 = 0$, so $p_1 = k$ (since f, g_1, and g_2 are not functions of θ_1). The Lie brackets with f are:

$$\begin{aligned} [f, g_1] &= \begin{bmatrix} \frac{\mu}{b_1^2}\frac{\partial b_1}{\partial \theta_2} \\ 0 \\ 0 \end{bmatrix} \\ [f, g_2] &= \begin{bmatrix} \frac{\mu}{b_1^2}\frac{\partial b_1}{\partial \theta_3} \\ 0 \\ 0 \end{bmatrix} \end{aligned} \tag{3.23}$$

$[g_1, g_2]$ also lies in the $\dot{\theta}_1$ direction, as shown in Equation (3.7), so the equation for the controls becomes:

$$\begin{aligned} \begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \end{bmatrix} &= \begin{bmatrix} \frac{k}{b_1^2}\left(b_3\frac{\partial b_1}{\partial \theta_2} - b_1\frac{\partial b_3}{\partial \theta_2} - b_2\frac{\partial b_1}{\partial \theta_3} + b_1\frac{\partial b_2}{\partial \theta_3}\right)u_2 \\ -\frac{k}{b_1^2}\left(b_3\frac{\partial b_1}{\partial \theta_2} - b_1\frac{\partial b_3}{\partial \theta_2} - b_2\frac{\partial b_1}{\partial \theta_3} + b_1\frac{\partial b_2}{\partial \theta_3}\right)u_1 \end{bmatrix} - \begin{bmatrix} \frac{k\mu}{b_1^2}\frac{\partial b_1}{\partial \theta_2} \\ \frac{k\mu}{b_1^2}\frac{\partial b_1}{\partial \theta_3} \end{bmatrix} \\ &= \frac{k}{b_1^2}\begin{bmatrix} \frac{\partial b_1}{\partial \theta_2}(b_3u_2 - \mu) - (\frac{\partial b_1}{\partial \theta_3}b_2 + \frac{\partial b_3}{\partial \theta_2}b_1 - \frac{\partial b_2}{\partial \theta_3}b_1)u_2 \\ \frac{\partial b_1}{\partial \theta_3}(b_2u_1 - \mu) - (\frac{\partial b_1}{\partial \theta_2}b_3 + \frac{\partial b_2}{\partial \theta_3}b_1 - \frac{\partial b_3}{\partial \theta_2}b_1)u_1 \end{bmatrix} \end{aligned} \tag{3.24}$$

with

$$\begin{aligned} \dot{\theta} &= f(\theta) + g_1(\theta)u_1 + g_2(\theta)u_2 \\ \dot{p} &= \frac{k}{b_1^2}\begin{bmatrix} 0 \\ \frac{\partial b_1}{\partial \theta_2}(\mu - b_2u_1 - b_3u_2) + \frac{\partial b_2}{\partial \theta_2}b_1u_1 + \frac{\partial b_3}{\partial \theta_2}b_1u_2 \\ \frac{\partial b_1}{\partial \theta_3}(\mu - b_2u_1 - b_3u_2) + \frac{\partial b_2}{\partial \theta_3}b_1u_1 + \frac{\partial b_3}{\partial \theta_3}b_1u_2 \end{bmatrix} \\ u_i &= -p^T g_i(\theta) \end{aligned} \tag{3.25}$$

Analytic approaches such as sinusoidal steering with the chained form or optimal control have the drawback that, because of their complexity, the controls generated may not provide any insight into the structure of the system. A further drawback is that these methods require a full model of the controlled system to be known, a requirement that becomes more onerous as the system becomes more complex, as is the full, three-dimensional diver model. Finally, these methods do not address simultaneously the issues of required configurations during the dive (such as the pike position) and of a required amount of time for the dive execution. In systems with nonlinear drift, such as the diver, the controls cannot simply be scaled to execute the movement in varying amounts of time. For the optimal control approach, the amount of time is specified, but to include required intermediate configurations, the time at which they are to be achieved must be known.

3.2.3 Behavioral Approaches

Another approach to controlling a human performing a dive or other motor task is the state machine method used in three-dimensional dynamic animation by Hodgins [41] for running and by Wooten and Hodgins [110] for diving. Their system is based on the scheme used by Raibert for running and hopping robots mentioned in Chapter 1 [80], [79]. The dive or motor task is divided into segments (similar to the behaviors used here) which form the machine states; within a state, the joints are moved to a goal point with PD controllers. Though they have achieved good results, especially when viewed as an animation task rather than a control task, there are several drawbacks with their control method. Because all the movements are produced with PD controllers, the motion is entirely feedback in origin, which is unrealistic from a biological point of view, and may be partly to blame for the slightly unnatural appearance of their running animations. Also, there are many parameters in the controllers that need to be tweaked by hand, though recently they have begun to apply simulated annealing methods to selecting the best parameter values for new animated characters given the tuned values for a different character [42].

Another approach using an idea of a behavioral architecture is that of Brooks [20], [18], who designed a robotic controller based on a subsumption architecture, in which each lower level of behavior is subsumed into the next higher one. For example, a low-level "stand up" behavior becomes part of the higher-level "simple walk" behavior [18]; signals from the walk controller can replace internal signals in the standup machine to modify its behavior to suit the higher-level goal. The controllers do not involve any dynamics; rather, they are based on augmented finite state machines whose transitions can depend on an input event or on a combinatorial predicate of the inputs. These controllers have been successfully applied to a series of robots, most notably insectoid walking robots. The subsumption controller is, in general, completely designed by hand; there is no learning involved. There have been a few experiments in learning done by Brooks' group (see [19]), in particular one in which an insectoid robot learned which behaviors to activate under which sensory conditions to achieve forward walking [61], but, in general, the approach has been to "evolve" the architecture by hand for the specific task.

Brockett's motion control language [14], [15], in which a controller specifies a feedforward control $\mathbf{u}$ (which can be an index to a family of controls), a feedback matrix $\mathbf{K}$, and a time T for which the control pair should be active, can also be seen as a behavioral control structure. Symbolic, hybrid approaches based on this description have been used by many roboticists (see Brockett's own paper [16] or [62], for example).

3.3 A Simple Learning Algorithm

A learning algorithm based on a behavioral control structure as described in Chapter 1 proved more promising than the analytical approaches for the planar diver ([24]). Even a very simple learning algorithm can, if given a suitably structured space to search, find a control law that will satisfy the control goal of driving the diver through a one-and-one-half-somersault pike. This task can be viewed as the coordination and timing of two multi-joint movements: entering the pike position and the exiting the pike into the layout position. Each of these multi-joint movements can, in turn, be viewed as a coupling of single-joint movements. Thus, the structure of the control family for the kinematic diving task was chosen based on the velocity profiles typically observed in fast, single-joint movements. In this type of movements, the limb involved typically has a single-peak velocity profile, as discussed in Section 2.5; a Gaussian is a good approximation to this curve [47]. The control structure used in the simple learning algorithm is thus based on parametrized Gaussian velocity profiles:

$$\begin{aligned} u_1 &= \frac{A}{\sqrt{2\pi\sigma^2}}\left[e^{-\frac{(t-\tau_1)^2}{2\sigma^2}} - e^{-\frac{(t-\tau_2)^2}{2\sigma^2}}\right] \\ u_2 &= -\frac{A}{\sqrt{2\pi\sigma^2}}\left[e^{-\frac{(t-\tau_1)^2}{2\sigma^2}} - e^{-\frac{(t-\tau_2)^2}{2\sigma^2}}\right]. \end{aligned} \tag{3.26}$$

The parameters that can be varied are given by

$$\mathbf{p} = \begin{bmatrix} A \\ \sigma^2 \\ \tau_1 \\ \tau_2 - \tau_1 \end{bmatrix} \tag{3.27}$$

where A is in radians per second and σ, τ_1, and τ_2 are in seconds. Each entry in $\mathbf{p}$ was restricted to a certain range; $\mathbf{p}_{\text{max}}$ and $\mathbf{p}_{\text{min}}$ are vectors of the maximum and minimum allowed values, respectively.

To choose the best value for the parameter vector $\mathbf{p}$, an algorithm based on gradient descent was used. The error on a dive was defined as

$$E(\mathbf{p}) = (\theta_1 - 3\pi)^2 + \theta_2^2 + (\theta_3 - \pi)^2 + k(\tau_2 + \sigma)^2 \tag{3.28}$$

The last term, in which k is a constant, provides a penalty for pulling out of the pike too late, since finishing the piked rotations early is considered good diving style. There are thus four constraints to minimize, and four parameters to learn. The gradient descent was preceded by N iterations with random parameters, from which the best parameter set was chosen to initialize the descent. Each of the four parameters was uniformly distributed within its allowed range. This technique served to start the descent algorithm in a favorable region of the control space, thus shortening the training time and ameliorating the problem of local minima to some extent. The algorithm began with:

$$\mathbf{p_1} = \text{random} \tag{3.29}$$

$$E_1 = E(\mathbf{p_1}) \tag{3.30}$$

Then, for each $i < N$:

$$\tilde{\mathbf{p}}_{\mathbf{i+1}} = \text{random} \tag{3.31}$$

$$\mathbf{p_{i+1}} = \text{best}(\mathbf{p_i}, \tilde{\mathbf{p}}_{\mathbf{i+1}}) \tag{3.32}$$

$$E_{i+1} = E(\mathbf{p_{i+1}}) \tag{3.33}$$

where $\text{best}(\mathbf{p_i}, \tilde{\mathbf{p}}_{\mathbf{i+1}})$ was determined by which vector had the lower error measurement. For $i \geq N$, the algorithm performed gradient descent with an estimated gradient $\mathbf{S}$:

$$\mathbf{S_N} = 0 \tag{3.34}$$

$$\mathbf{D_{i+1}} = \frac{(E_i - E_{i-1})(\mathbf{p_i} - \mathbf{p_{i-1}})}{||\mathbf{p_i} - \mathbf{p_{i-1}}||^2} \tag{3.35}$$

$$\mathbf{S_{i+1}} = \frac{m\mathbf{S_i} + \mathbf{D_{i+1}}}{m+1} \tag{3.36}$$

$$\tilde{\mathbf{p}}_{\mathbf{i+1}} = \mathbf{p_i} - \frac{E_i \mathbf{S}^{\mathbf{T}}_{\mathbf{i+1}} \mathbf{W}}{||\mathbf{S_{i+1}}||} + \text{noise} \tag{3.37}$$

$$\mathbf{p_{i+1}} = \text{best}(\mathbf{p_i}, \tilde{\mathbf{p}}_{\mathbf{i+1}}) \tag{3.38}$$

$$E_{i+1} = E(\mathbf{p_{i+1}}) \tag{3.39}$$

$\mathbf{S}$ is a weighted average of past gradient measurements $\mathbf{D}$, with more recent measurements weighted more heavily so the algorithm can adapt to different regions of the control space. m is thus a parameter describing how long the memory of the estimate is. Each entry in $\mathbf{S}$ was restricted to $[-3000, 3000]$. $\mathbf{W}$ is a vector of scaling factors; $\mathbf{W} = (\mathbf{p_{max}} - \mathbf{p_{min}})/40000$. Some uniformly distributed noise was added to the next parameter choice to avoid getting stuck at local minima or at a parameter maximum or minimum. The noise amplitude was proportional to E_i and $\mathbf{W}$. If E_i and $\mathbf{p}_i$ stayed the same for too long, the constant of proportionality was increased for one step, and in (3.38) $\mathbf{p}_{i+1}$ was always set equal to $\tilde{\mathbf{p}}_{i+1}$ in order to move to a different region of the control space. When the error measurement dropped below a cutoff value, the algorithm halted. In the simulations shown in Figures 3.2 and 3.3, $N = 20$, $m = 5$, $k = .0076$,

$$\mathbf{p}_{\min} = \begin{bmatrix} 0 \\ 0.004 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{p}_{\max} = \begin{bmatrix} 2.1 \\ 0.4 \\ 0.4 \\ 1.5 \end{bmatrix},$$

and the error cutoff was .00947. Since the gradient descent starts from the best of several random iterations, the number of steps the algorithm takes to terminate varies widely.

A simulation of the dive produced with the learned parameters is shown in Figures 3.2 and 3.3. The simple learning algorithm presented here generates movements that are qualitatively similar to those of human divers performing piked somersaults. The controls were kinematic, for better comparison to the above, more traditional control methods; the torques required to produce the controls (see Figure 3.2C) are the same order of magnitude as torques humans can produce. The success of this kinematic behavioral control structure suggested extending the idea to a more general, dynamic controller which would be able to learn to control the fully three-dimensional diver with many degrees of freedom; this is the learning controller presented in Chapter 4.

3.4 Three-Dimensional Diver Model

The three-dimensional diver model used in subsequent chapters has a large number of potential degrees of freedom, but for simplicity the simulations only make use of ten degrees of freedom in the joints: flexion at each hip and elbow and three degrees of freedom at each shoulder (modeled with ZYZ Euler angles). The body orientation is modeled with quaternions, which are converted into YXZ Euler angles to observe the tumbling rotations. A graphical representation of the model in the zero configuration is shown in Figure 3.4, and the physical parameters of the model are shown in Table 3.2. The model and simulations are implemented with the Symbolic Dynamics, Inc.'s SD/FAST software [45] (which uses Kane's

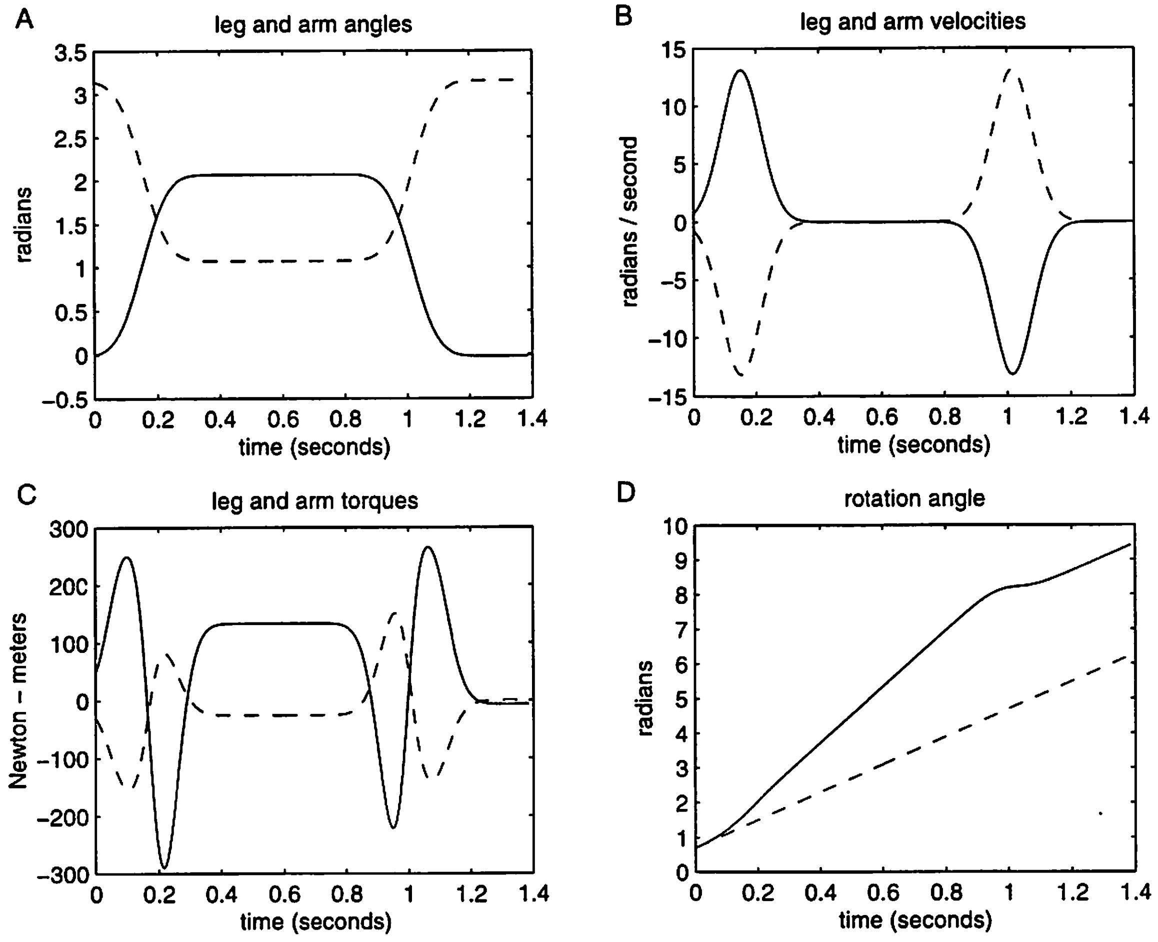


Figure 3.2: Simulation with control parameters chosen by the learning algorithm. $A = 2.084866$, $\sigma^2 = 0.004000$, $\tau_1 = 0.152110$, and $\tau_2 - \tau_1 = 0.863879$. A. θ_2 (solid) and θ_3 (dashed). B. $\dot{\theta}_2$ (solid) and $\dot{\theta}_3$ (dashed). C. Leg (solid) and arm (dashed) torques required to produce the movement. D. θ_1 with these controls (solid) and with the controls set to zero (dashed).

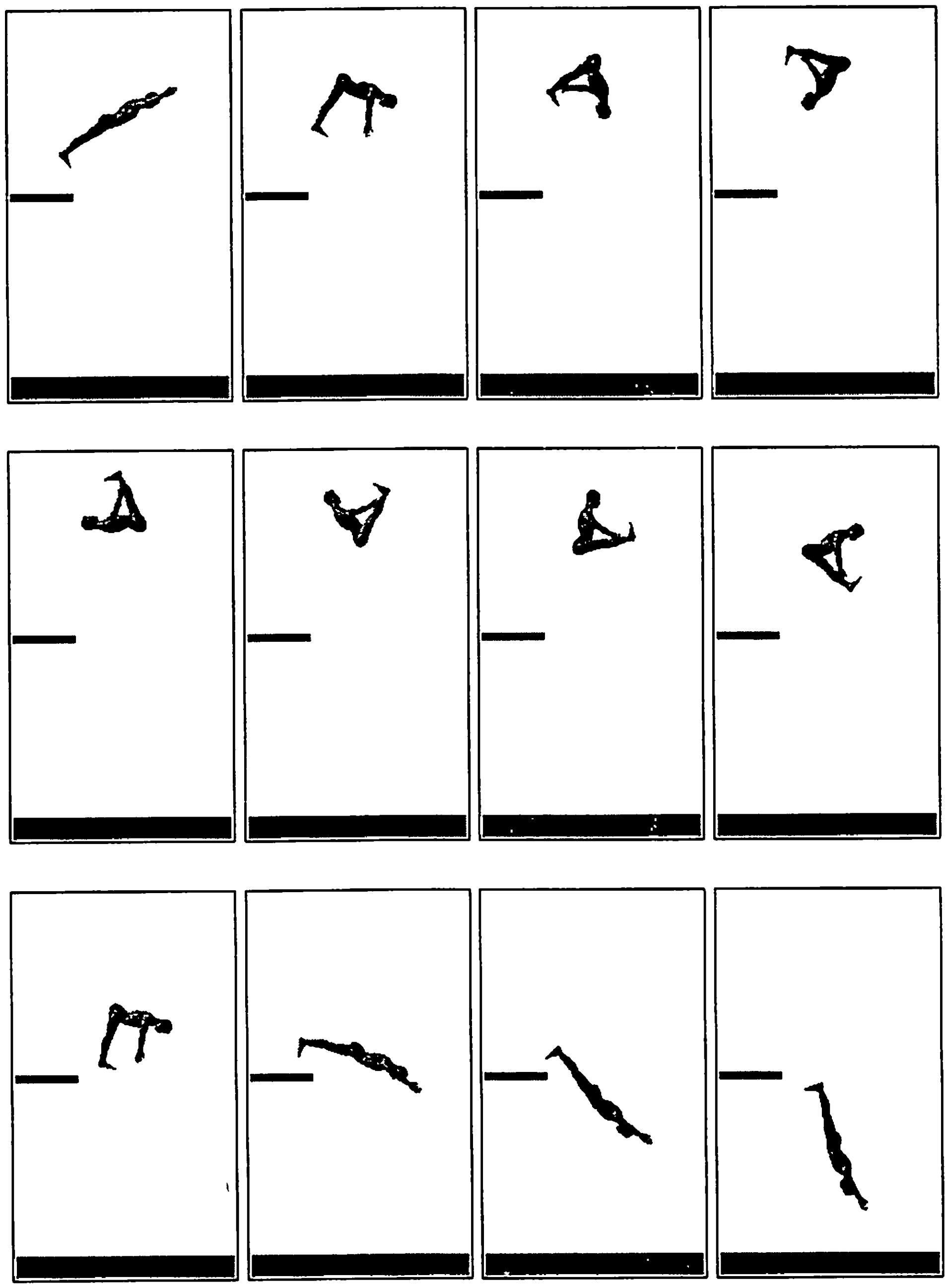


Figure 3.3: Frames from the simulation shown in Figure 3.2. The graphical human model is from Viewpoint DataLabs.

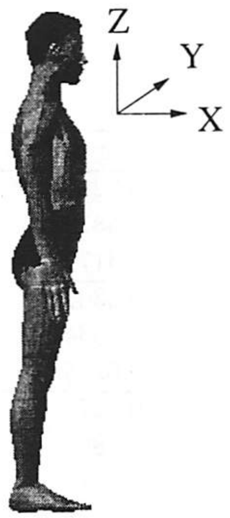


Figure 3.4: Three-dimensional diver model in the zero configuration. The graphical model is from Viewpoint DataLabs.

algorithm for implementing the dynamics). The integration tolerance for the variable-step integrator is set to 10^{-6}.

In biological systems, joint receptors can signal the nervous system when the joint is nearing the limits of its range. In the diver simulation, these joint limits are modeled by stiff springs and dampers. The joint ranges used are shown in Table 3.3. These limits keep the shoulders within the range where the Euler angles are continuous. Note that the limits for joints 4 and 5 (the Y Euler angle) are a bit inside the full π range; this restriction is necessary to avoid the singularity in the Euler angles (because the boundary is modeled with springs and dampers, and is therefore somewhat soft). The spring and damper constants used for the joint limit are both 750; the torques produced by the joint limit servos are scaled according to the inertia of the limb in its current configuration by being passed through the inertial compensator described in Section 4.1.

Link Name	Mass (kg)	$\mathcal{I}$ (kg·m^2)	Center (m)	Joint (m)	DOF
		0.7286	0.0000	0.0000	
torso	29.2720	0.6319	0.0000	0.0000	6
		0.3172	0.0000	0.0000	
		0.0301	0.0216	0.0124	
head	5.8988	0.0334	0.0000	0.0000	0
		0.0228	0.3853	0.3213	
		0.2272	-0.0095	0.0124	
pelvis	16.6085	0.1809	0.0000	0.0000	0
		0.1575	-0.3248	-0.2217	
		0.1545	-0.0289	-0.0048	
thigh, right	8.3460	0.1591	-0.1056	-0.0990	1
		0.0256	-0.5559	-0.4357	
		0.1545	-0.0289	-0.0048	
thigh, left	8.3460	0.1591	0.1056	0.0990	1
		0.0256	-0.5559	-0.4357	
		0.0553	-0.0860	-0.0805	
calf, right	4.1617	0.0565	-0.1051	-0.0862	0
		0.0073	-0.9341	-0.7689	
		0.0553	-0.0860	-0.0805	
calf, left	4.1617	0.0565	0.1051	0.0862	0
		0.0073	-0.9341	-0.7689	
		0.0016	-0.0591	-0.0875	
heel, right	1.1703	0.0045	-0.1027	-0.0955	0
		0.0040	-1.2287	-1.1848	
		0.0016	-0.0591	-0.0875	
heel, left	1.1703	0.0045	0.1027	0.0955	0
		0.0040	-1.2287	-1.1848	
		0.0001	0.0693	0.0409	
toes, right	0.1689	0.0001	-0.1172	-0.0955	0
		0.0002	-1.2584	-1.2646	
		0.0001	0.0693	0.0409	
toes, left	0.1689	0.0001	0.1172	0.0955	0
		0.0002	-1.2584	-1.2646	

Link Name	Mass (kg)	$\mathcal{I}$ (kg·m^2)	Center (m)	Joint (m)	DOF
		0.0255	-0.0474	-0.0476	
upper arm, right	2.7943	0.0252	-0.2205	-0.1640	3
		0.0050	0.0098	0.1293	
		0.0255	-0.0474	-0.0476	
upper arm, left	2.7943	0.0252	0.2205	0.1640	3
		0.0050	0.0098	0.1293	
		0.0050	-0.0277	-0.0529	
lower arm, right	1.2164	0.0054	-0.2504	-0.2571	1
		0.0012	-0.2503	-0.1605	
		0.0050	-0.0277	-0.0529	
lower arm, left	1.2164	0.0054	0.2504	0.2571	1
		0.0012	-0.2503	-0.1605	
		0.0016	0.0102	-0.0156	
hand, right	0.5511	0.0020	-0.2641	-0.2644	0
		0.0005	-0.4485	-0.3636	
		0.0016	0.0102	-0.0156	
hand, left	0.5511	0.0020	0.2641	0.2644	0
		0.0005	-0.4485	-0.3636	

Table 3.2: Physical parameters for the full diver model, to four decimal places, showing the link name, mass in kilograms, inertia matrix (principal axes of all links are aligned with the global axes in the zero configuration) in kilogram meters squared, coordinates of the center of mass in meters, coordinates of the connecting joint in meters, and number of degrees of freedom in the diver simulation. This model was generously shared with us by Professor Hodgins at the Georgia Institute of Technology; her group determined these parameters from anatomical density data and the Viewpoint DataLabs graphical model as described in [110].

Joint Name	DOF #	Min Value	Max Value
Right hip	0	-3.1416	0.0000
Left hip	1	-3.1416	0.0000
Right shoulder	2	-3.1416	0.0000
	4	-3.0916	-0.0500
	6	-3.1416	3.1416
Left shoulder	3	0.0000	3.1416
	5	-3.0916	-0.0500
	7	-3.1416	3.1416
Right elbow	8	-3.1416	0.0000
Left elbow	9	-3.1416	0.0000

Table 3.3: Joint limits for the 10 degrees of freedom in the diver model, in radians.

Chapter 4

Learning Controller

There have been many learning controller designs proposed in the literature. These range from engineering adaptive control approaches [86] and iterative learning approaches for repetitive systems [69], [75] to detailed biological approaches such as the Adjustable Pattern Generators modeled on the cerebellum [9]. Many fall somewhere in between these, often with neural networks as part of the dynamic control structure, such as Nguyen and Widrow's truck backer-upper [74] or Kawato's feedback error learning scheme, in which the control changes from feedback to feedforward as the system learns [55] (see [2] for a review). Recursive learning based on continuous, parametrized controls has been used with radial basis functions to train a space robot with a nonholonomic arm [32], as well as with a Jacobian estimate to optimize the design of a mechatronic system [78]. Approaches based on learning through demonstration ("learning by watching") have been used with various control structures for diverse problems; recent examples include pole-balancing [88] and helicopter control [67].

The controller design presented here, shown in Figure 4.1, was inspired by the hierarchical, behavioral control structure used by biological systems to control their movement, as discussed in Chapter 2. The dives executable by the system are represented as a parametrized class; the parameters specifying a particular desired dive then form the input to the controller. The high-level coordinating controller coordinates between several multi-joint behavioral synergies, learning the timing between them (see Section 4.2). The multi-joint synergies themselves provide the tuning inputs for the low-level single-degree-of-freedom (single-DOF) controllers, each of which plays the role of a pattern generator for fast, single-joint movements (see Section 4.1). The coordinating controller and the single-

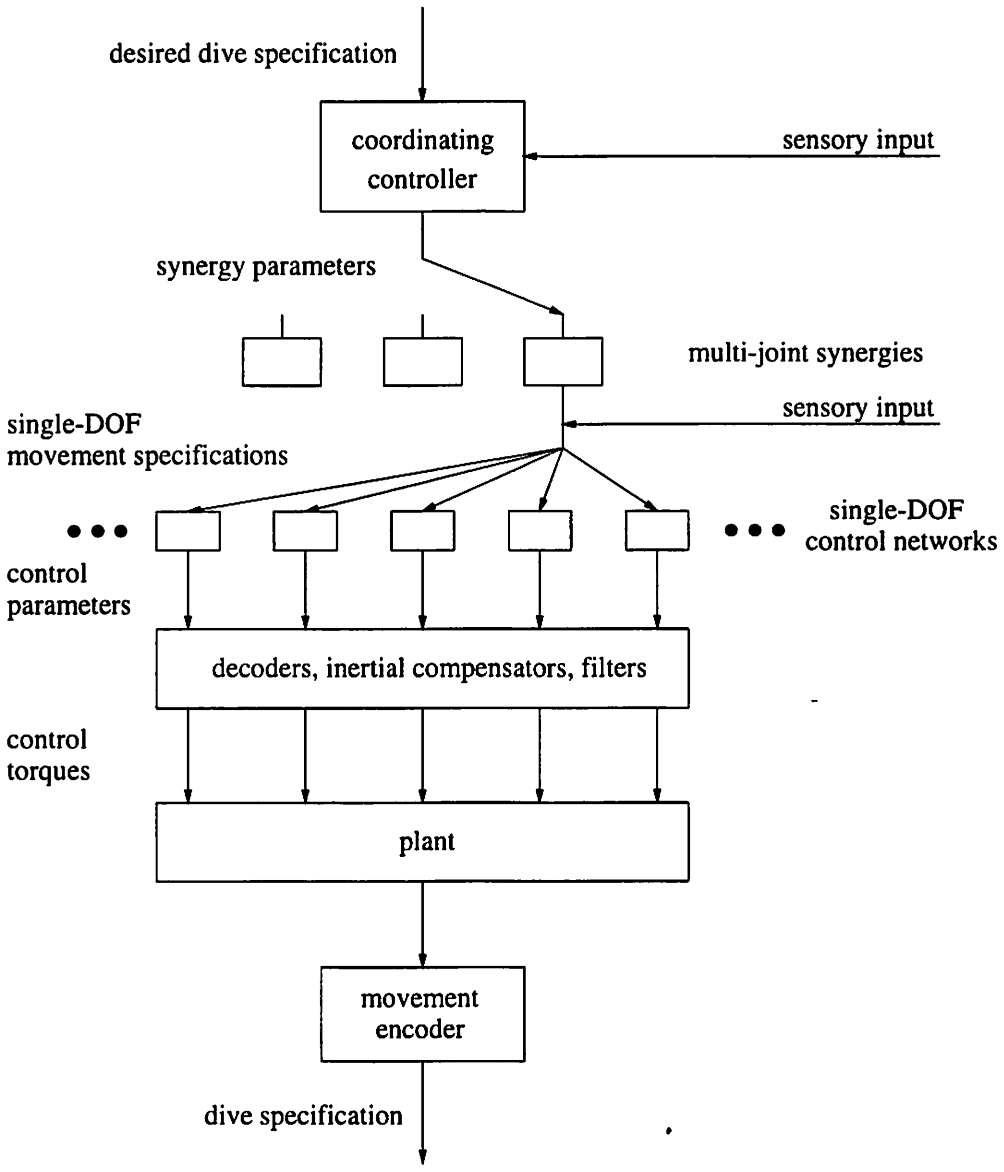


Figure 4.1: Overview of the hierarchical design for the learning controller.

DOF controllers all operate in the discrete time domain. The plant, a mechanical system, operates in continuous time; the pattern generator structure in the single-DOF controllers allows this simplification of the control representation and provides the link between the two regimes.

4.1 Single-DOF Controllers

4.1.1 Design

A closeup of a low-level controller is shown in Figure 4.2. This controller is inspired by Gottlieb, Corcos, and Agarwal's [34] model for a control method for fast, goal-directed, single-joint movements, which can be seen as a pattern generator for these kinds of movements, as discussed in Section 2.4. Because of the many degrees of freedom and because the desired dive is only specified by general, behavioral metrics, there may be many possible control torque profiles that would achieve the goal. A pattern generator is one way of restricting these control choices to torque profiles within a certain parametrized family, making the control problem easier. As shown in Figure 4.2, the single-DOF controller is made up of a control network, a decoder, an inertial compensator, and a filter.

The control network takes as input a vector $[\mathbf{y_d}, \dot{\theta}_0] = [\Delta\theta_d, \Delta\dot{\theta}_d, T_d, \dot{\theta}_0] \in \mathbb{R}^4$ describing the desired change in the DOF angle (a particular Euler angle, if the DOF is part of a multi-DOF joint), the desired change in joint velocity, the desired movement duration, and the current DOF velocity (sensory information available from the spindle organs in biological systems). The output of the network is a vector in $\mathbb{R}^3$ giving a parametrization of the control torque profile. The torque profile family used is a filtered version of a two-rectangular-pulse function, as shown in Figure 4.2. The control network thus provides the translation from kinematics to dynamics, all in a discrete time regime, abstracting the dynamics out of the control and learning problem; the controller network need not provide continuous-time torque information directly. By restricting the torque to a parametrized family of functions, the single-DOF controller defines a behavioral model for the allowed class of single-DOF motions.

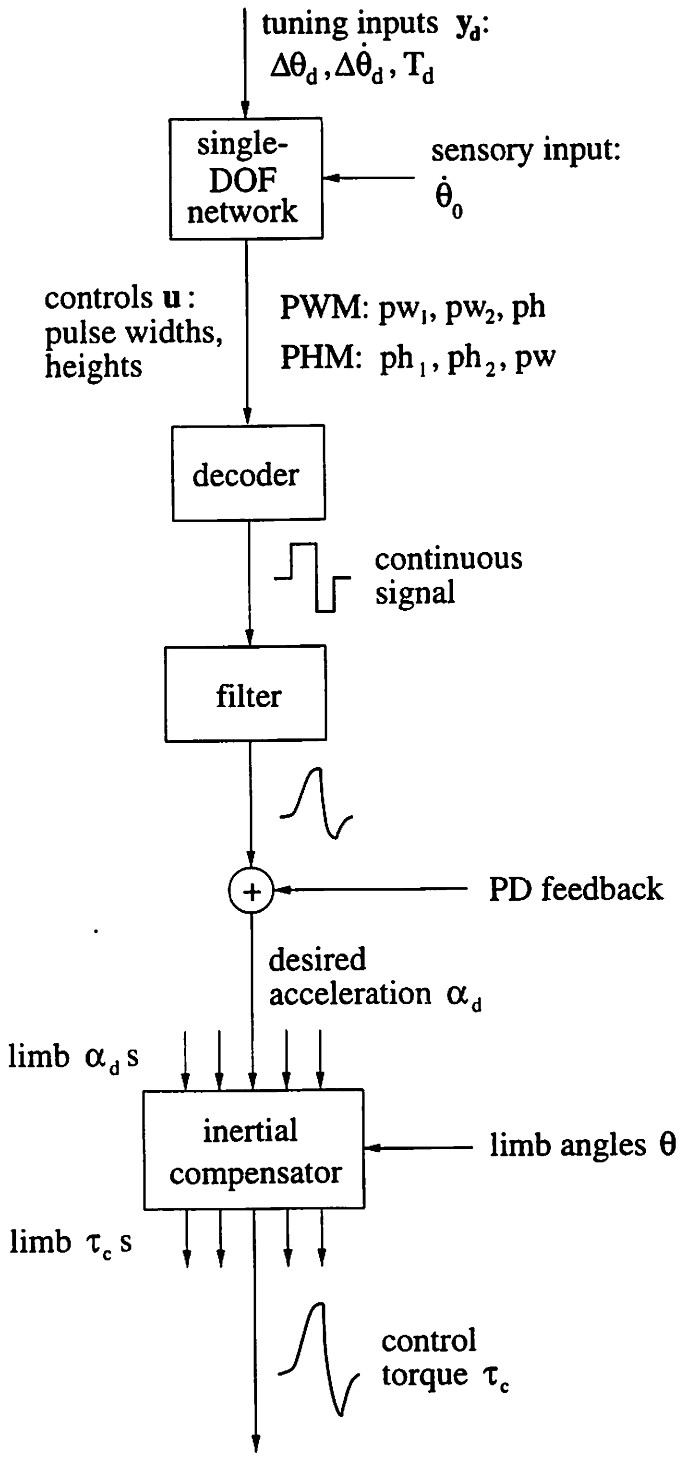


Figure 4.2: Design of the single-DOF controller.

Control Network and Decoder

Although the model for a control mechanism for biological fast, goal-directed, single-joint movements proposed by Gottlieb, Corcos, and Agarwal and their colleagues ([34], [33], [48]; see Section 2.4) describes the biological data fairly well, from the point of view of the single-DOF controller, it has a few drawbacks. First, it does not allow for a change in velocity from the beginning to the end of the movement; the limb velocity starts and ends at zero. Second, the model contains four control parameters: the pulse width W, the pulse heights H_{ag} and H_{ant}, and the antagonist latency T_{ant}, plus the inertia J which parametrizes the movement. These control parameters are functions of only three task variables, the extent X, the effort F, and the target distance D; thus, as control parameters produced by this model do not span the space of all possible controls, a randomly generated control vector cannot necessarily be produced by the model. Since the physical system used for the diver simulation is not identical to the human one to which Gottlieb fit his model [33], using his model and parameters directly will not necessarily result in reasonable movements that span the desired range of behavior. In order to fit the controls to the dynamics appropriately, though, some sort of data is required. Since simulating a random control vector will not guarantee a control/outcome pair that is a valid sample for the Gottlieb model, parameter fitting cannot be done with the diver system using this model.

The model adopted for the single-DOF controller is similar to the Gottlieb model but has some major differences. First, as mentioned above, there are three controls for three task variables, $\mathbf{y} = [\theta, \dot{\theta}, T]$. There is no latency in the onset of the antagonist activation pulse; it begins as soon as the agonist pulse ends. The controller uses two modes of operation, pulse-width modulation (PWM) and pulse-height modulation (PHM). PWM mode restricts the heights of the agonist and antagonist pulses to be the same, while PHM mode restricts the widths of the two pulses to be the same. The single-DOF controller chooses between the two modes based on the desired duration of the movement; if the movement time is above a cutoff, then PWM is used, and if it is below, PHM is used. Thus, for shorter movements when less time is available for pulse width variation to be effective, the controller has access to varying pulse height to achieve a wide range of movements, while for longer movements, pulse width variation makes using large, variable pulse heights unnecessary. This division is similar to Gottlieb's observation, discussed in Section 2.4, that different control strategies are used for small and large X; for $X < X_0$, the pulse heights are roughly proportional to X, and for $X > X_0$, the pulse heights are roughly constant with respect to X (see Equations (2.1)). As mentioned in Section 2.4, Gottlieb attributes this effect to the existence of a physiological minimum achievable pulse width; for movements over short distances it is necessary to vary the pulse heights to make a range of movements possible. In Gottlieb's model, of course, the two pulses are always constrained to have the same width. Samples of movements produced by the trained controller, demonstrating the two-pulse controls as well as the bell-shaped velocity profiles, are shown in Figures C.7–C.5.

In PWM mode, the control vector output of the network gives the pulse width of each of the two pulses, and the common pulse height of both; $\mathbf{u} = [\mathrm{pw}_1, \mathrm{pw}_2, \mathrm{ph}] \in \mathbb{R}^3$. In PHM mode, it gives the two pulse heights and the common pulse width; $\mathbf{u} = [\mathrm{ph}_1, \mathrm{ph}_2, \mathrm{pw}] \in \mathbb{R}^3$. The decoder translates this control vector into a two-pulse acceleration trajectory, a continuous time signal. Although in nature these stereotypical torque profiles may be produced by a combination of descending control and reciprocal inhibition, as discussed in Section 2.4, here all these interactions are encapsulated in one feedforward control plus feedback around the desired trajectory to compensate for external torques and joint interactions not taken into account by the single-DOF controller (see below). In both PWM and PHM modes, the control representation $\mathbf{u}$ which will produce a particular $\mathbf{y}_\mathrm{d}$ is unique given a particular $\dot{\theta}_0$. This uniqueness can be verified algebraically for unfiltered pulses, or by examining data sets produced using filtered pulses.

Filter and Inertial Compensator

The filter is analogous to the smoothing effect of the motoneuron pool and the muscles, as discussed in [34]. The linear filter used in the single-DOF controller is given by the transfer function

$$H(s) = \frac{\alpha\beta}{(s+\alpha)(s+\beta)} \tag{4.1}$$

$\alpha = 65.0$, $\beta = 60.0$ were chosen to provide the desired control flexibility and to give a shape and rise time roughly similar to torque data from human arm movements. These filtered pulses form a feed-forward acceleration signal, which will be scaled to become the feed-forward torque.

The inertial compensator computes the moment of inertia for each DOF based on the current positions of the joints and scales all the torque profiles accordingly. The calculations are simplified considerably by an approximation restricting the equations to joints on the same limb; the body to which the limb attaches is considered to be a fixed base, and the other three limbs are ignored. In theory, the function of the compensator could be included in the control network, but in practice, this would require adding too many inputs (the current angles of all relevant degrees of freedom), as well as more hidden units, and the network would be quite difficult to train. The equations used for calculating the mass matrices as functions of the joint angles on a given limb (see [70]) are given by

$$\mathbf{M}(\theta) = \sum_{i=1}^{n} \mathbf{J}_{\mathrm{i}}(\theta)^{\mathrm{T}} \mathcal{M}_{\mathrm{i}} \mathbf{J}_{\mathrm{i}}(\theta) \tag{4.2}$$

where the $\mathbf{J}_{\mathrm{i}}$s are the body Jacobians for each of the n controlled degrees of freedom on the limb and θ is a vector of the angles of the degrees of freedom. The $\mathcal{M}_{\mathrm{i}}$s are the generalized inertia matrices for each link, which depend on both the link masses m_i and their inertia matrices $\mathcal{I}_i$:

$$\mathcal{M}_{\mathrm{i}} = \begin{bmatrix} m_i\mathbf{I} & 0 \\ 0 & \mathcal{I}_i \end{bmatrix} \tag{4.3}$$

The torques output from the compensator for the limb, τ_c, are then obtained from the vector of desired accelerations, α_d, coming from the summing junctions of all the single-DOF controllers of the limb (see Figure 4.2):

$$\tau_c = \mathbf{M}(\theta)\alpha_d \tag{4.4}$$

A more detailed discussion of the inertial compensator equations is given in Appendix A.

All perturbations, multi-joint interactions not taken care of by the inertial compensator (such as centripetal and Coriolis terms), gravity, and external torques are compensated for by a feedback term. The filtered decoder signal specifies the desired acceleration of the joint; using this signal, the current desired position and velocity are calculated and used for PD feedback. In a biological system, the actual position and velocity would be available from the spindle fiber sensory organs for use in feedback (see Chapter 2). The PD constants used were $k_p = k_d = 75.0$ for all degrees of freedom. The feedback signal is added to the feedforward signal before they are passed through the inertial compensator.

4.1.2 Learning

As the control network is simply required to represent a static function $f : \mathbb{R}^4 \rightarrow \mathbb{R}^3$, any of a number of modeling choices are possible, including neural networks (multi-layer perceptrons) as well as approximators that are linear in their parameters, such as radial basis function networks, wavelets, infinite support basis function representations like Fourier series, and polynomial approximation. Neural networks have several advantages over the other choices. The number of parameters (weights) needed does not grow exponentially with the number of inputs, unlike with the other methods, providing a compact representation as well as fewer data points needed for parameter identification. The smooth, global approximation provided by neural networks is a better representation for the functions being approximated here than that given by the methods with local-support basis functions, such as radial basis functions or wavelets [95]. The nonlinearities in the function that is being approximated are much more easily approximated by neural networks than by polynomial approximation. One major drawback of neural networks as compared to approximators that are linear in their parameters is that training methods can only be guaranteed to find local minima; in training the single-DOF controllers, however, this has not proved to be a problem.

One of the most basic kinds of neural network, and the kind used in this work, is a multilayer perceptron. This type of network consists of several layers of nodes, or units, each layer communicating with the nodes of the previous layer (or, sometimes, layers) through weighted connections. The first layer of the network is called the input layer and the final layer is the output layer; all the other layers are referred to as hidden layers. Each node (in each layer other than the input layer) i receives a net input n_i that is a linear combination of the outputs, or activations a_j, of all the nodes j it is connected to in the previous layer, plus an internal bias. The coefficients of the linear combination are the network weights, w_{ij}. To simplify the equations, the bias can be represented as a connection from a node whose activation is always 1, and therefore can be treated in the same manner as the other weights. In the most standard architecture, a node is connected to all nodes in the previous layer. Thus, for a node i, and nodes j in the previous layer:

$$n_i = \sum_j w_{ij} a_j \tag{4.5}$$

The net input of a node is passed through an activation function to determine the activation of that node. Typical activation functions include a simple thresholding function ($a_i = 1$ if $n_i > \theta$, otherwise $a_i = 0$), a linear function, or a sigmoidal function. The networks described here use a linear function ($a_i = n_i$) for the output nodes of the forward networks (see below) and a sigmoidal function

$$a_i = \frac{1}{1+e^{-\beta n_i}}, \tag{4.6}$$

with $\beta = .8$, for all other units.

Network Training

There are several ways to go about training a network for control. Agarwal [2] presents a classification of the way neural networks are used in control. A brief, less general discussion can be found in the work of Werbos [106]. In the network design presented here, the input and output spaces are relatively low-dimensional, and the network is simply learning a static function, so some form of supervised learning is the simplest choice. From there, there are two possibilities for the training procedure: direct learning or indirect learning. From an adaptive control viewpoint, these two choices correspond to the choice between model reference adaptive control (direct) and a self-tuning regulator (indirect) [86]. In direct learning, the simpler of the two, the network is trained on pairs $([\mathbf{y}_\mathrm{d}, \dot{\theta}_0], \mathbf{u})$ of plant outputs and the control inputs that produce them. The main drawback of this method is that the error that is minimized is the error on the controls $\mathbf{u}$, when what is desired is to minimize the error on the final plant output $\mathbf{y} = [\Delta\theta, \Delta\dot{\theta}, T] \in \mathbb{R}^3$ in response to those controls. Indirect learning, on the other hand, minimizes the error on $\mathbf{y}$, but requires a second network to approximate the action of the plant; this forward network must approximate a function $f' : \mathbb{R}^4 \rightarrow \mathbb{R}^3$ mapping $[\mathbf{u}, \dot{\theta}_0]$ to $\mathbf{y}$. The forward network is trained first on $([\mathbf{u}, \dot{\theta}_0], \mathbf{y})$ pairs and is then used in the training of the inverse network. During training of the inverse network, the error at the output of the forward, plant-approximating network is propagated back through that network without modifying those weights, and the resulting back-propagated error is used to train the inverse network. Using indirect learning has another advantage: once the networks have been trained, the forward network can be used to test whether a particular command $\mathbf{y}_\mathrm{d}$ (with the current DOF velocity $\dot{\theta}_0$) is possible given the range of controls available (see Figure 4.3). If the control $\mathbf{u}$ generated by the inverse network is passed through the forward network to produce a predicted $\hat{\mathbf{y}}$, but $||\hat{\mathbf{y}} - \mathbf{y}_\mathrm{d}||$ is large, the requested $\mathbf{y}_\mathrm{d}$ is likely to be outside the range of the available controls. This consideration is important in the design presented here, where the boundaries of the output space reachable by the available controls are not known, but the reachable space is known

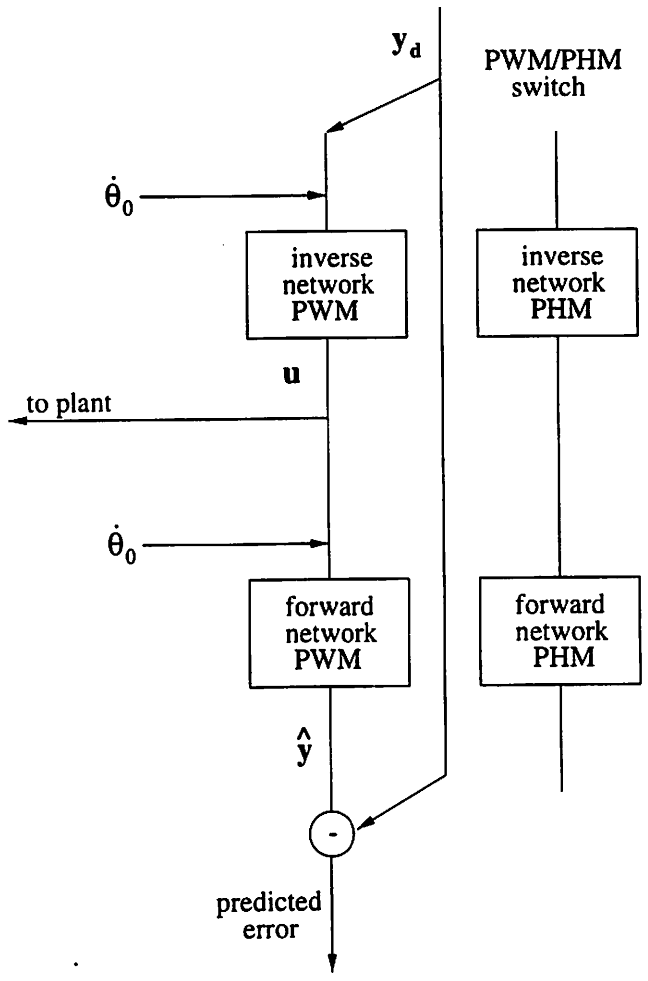


Figure 4.3: Diagram showing the relationships between the four trained networks that make up the network controller.

to exclude a significant proportion of the possible $[\mathbf{y}_d, \dot{\theta}_0]$. All of these considerations led toward the choice of an indirect learning method.

Feedforward neural networks of this type have been shown, by various methods, to be capable of approximating a large class of continuous functions from one finite-dimensional space to another, as well as decision boundary functions, to arbitrary accuracy, given one hidden layer of unlimited size (see [12], [39] for discussion and lists of references). In addition, one-sided inverses of continuous functions can be approximated to arbitrary accuracy with networks with two hidden layers and step activation functions [94]. There are some rules of thumb available for the required size of the hidden layers with respect to the number of data points and the approximation error (see [38], Chapter 6, for example), but this choice is still often a matter of art more than science. In practice, hidden layers with more units make for slower and more difficult training. By experimentation, it was determined that the forward networks could be adequately trained if they had two hidden layers of eight nodes each; the inverse networks required two hidden layers of twelve nodes each. A single hidden layer did not train as well. Adding more hidden nodes failed to increase the accuracy of the approximation considerably, trained more slowly, and required a larger training data set to generalize as well.

The total network error is given by the sum of the squared errors over all patterns and all outputs:

$$E = \sum_p \sum_m \frac{1}{2}(o_{p,m} - t_{p,m})^2 =: \sum_p \sum_m \frac{1}{2} e_{p,m}^2 =: \sum_p E_p \tag{4.7}$$

where the subscript p indicates the training pattern, the subscript m indicates the network output unit, $t_{p,m}$ is the target for output m in pattern p, $o_{p,m}$ is the actual output for output unit m on pattern p, and $e_{p,m}$ is the output error for output unit m on pattern p. To find the optimal network weights, this squared error equation must be minimized. There is a family of methods available for least-squares minimization. Linear least-squares methods, for systems in which the error e is linear in the approximator parameters, include the traditional linear least squares using singular value decomposition as a batch method, and recursive least squares, or the Kalman filter, an iterative method often used in adaptive control. For nonlinear systems, such as neural networks, various gradient methods are available, including basic gradient descent (either as a batch or iterative method), conjugate gradients (a batch method), Newton's method, quasi-Newton methods, and the Gauss-Newton method (batch methods), and the extended Kalman filter (an iterative method similar to the Gauss-Newton method). Many of these methods (the Kalman filter and extended Kalman filter, gradient descent, the Newton family) are similar in that at each iteration, the approximator weights or parameters are updated along a direction $-\mathbf{G}\mathbf{g}^{\mathrm{T}}$, where $\mathbf{g}$ is the gradient of E (or E_p, for iterative methods) with respect to the weights and $\mathbf{G}$ is a matrix that modifies the direction of the gradient. In gradient descent, $\mathbf{G}$ is the identity. In Newton's method, $\mathbf{G}$ is the inverse Hessian of E with respect to the weights. In quasi-Newton methods, the Gauss-Newton method, and the extended Kalman filter, $\mathbf{G}$ is an approximation of the Hessian inverse, as follows. In the linear problem,

$$E = \sum_p \frac{1}{2} ||\hat{\mathbf{y}}_{\mathbf{p}} - \mathbf{y}_{\mathbf{p,d}}||^2 = \sum_p \frac{1}{2} ||\phi_{\mathbf{p}}^{\mathbf{T}} \mathbf{w} - \mathbf{y}_{\mathbf{p,d}}||^2 \tag{4.8}$$

where ϕ_p is a matrix whose columns are called regressor vectors. In this problem, the Hessian is given by

$$\mathbf{H} = \sum_p \phi_\mathrm{p}\phi_\mathrm{p}^\mathrm{T} \tag{4.9}$$

In the Gauss-Newton method, the nonlinear errors $e_p = \hat{\mathbf{y}}_\mathrm{p} - \mathbf{y}_{\mathrm{p,d}}$ are linearized around the current parameter vector, yielding approximate regressor matrices $\hat{\phi}_\mathrm{p}$ equal to the transpose of the gradient of this error. Then, the inverse Hessian is approximated as $\left(\sum_p \hat{\phi}_\mathrm{p}\hat{\phi}_\mathrm{p}^\mathrm{T}\right)^{-1}$. The extended Kalman filter is an iterative version of this linearization method which maintains an approximation of the inverse Hessian called the covariance matrix. The quasi-Newton family of methods, though batch methods, incrementally build up a more complex approximation of the inverse Hessian over successive iterations, as will be described below. More detailed discussions of these methods can be found in [11], [12], [60], and [39]. For continuous time Kalman filtering in an adaptive control context, see [86].

Quasi-Newton Algorithm

For training the single-DOF networks, I have chosen a quasi-Newton algorithm [12], [60]. Quasi-Newton algorithms are useful because, like Newton's method, they tend to be faster than gradient descent, but unlike Newton's method, they do not require the calculation and inversion of the Hessian matrix. Instead, as discussed above, quasi-Newton methods maintain and update an estimate $\mathbf{G}$ of the inverse of the Hessian.

With a locally quadratic approximation to the error surface as a function of the network weights, the Hessian $\mathbf{H}$ is constant, and one obtains

$$\mathbf{g}(\mathbf{w}) = (\mathbf{w} - \mathbf{w}^*)^\mathrm{T}\mathbf{H}, \tag{4.10}$$

where $\mathbf{g}(\mathbf{w})$ is the gradient at a particular weight vector $\mathbf{w}$, $\mathbf{w}^*$ is the weight vector at the error minimum, and $\mathbf{H}$ is the Hessian evaluated at $\mathbf{w}^*$. To jump immediately to the minimum at $\mathbf{w}^*$, as in Newton's method, one would simply use

$$\mathbf{w}^* = \mathbf{w} - \mathbf{H}^{-1}\mathbf{g}^\mathrm{T}(\mathbf{w}) \tag{4.11}$$

From these equation one obtains what is known as the quasi-Newton condition on successive weights:

$$\mathbf{w}^{k+1} - \mathbf{w}^k = -\mathbf{H}^{-1}((\mathbf{g}^{k+1})^\mathrm{T} - (\mathbf{g}^k)^\mathrm{T}), \tag{4.12}$$

where $\mathbf{g}^k = \mathbf{g}(\mathbf{w}^k)$. A quasi-Newton algorithm is one in which the approximation $\mathbf{G}$ of the inverse Hessian fulfills this condition on the weight increments. The Broyden-Fletcher-Goldfarb-Shanno (BFGS) procedure is a quasi-Newton algorithm with update rules:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \alpha^k \mathbf{G}^k (\mathbf{g}^k)^{\mathrm{T}} \tag{4.13}$$

$$\begin{aligned} \mathbf{G}^{k+1} &= \mathbf{G}^k + \frac{\mathbf{p}\mathbf{p}^{\mathrm{T}}}{\mathbf{p}^{\mathrm{T}}\mathbf{v}} - \frac{(\mathbf{G}^k\mathbf{v})(\mathbf{v}^{\mathrm{T}}\mathbf{G}^k)}{\mathbf{v}^{\mathrm{T}}\mathbf{G}^k\mathbf{v}} + \mathbf{v}^{\mathrm{T}}\mathbf{G}^k\mathbf{v}\mathbf{u}\mathbf{u}^{\mathrm{T}} \qquad (4.14) \\ \mathbf{p} &= \mathbf{w}^{k+1} - \mathbf{w}^k \\ \mathbf{v} &= (\mathbf{g}^{k+1})^{\mathrm{T}} - (\mathbf{g}^k)^{\mathrm{T}} \\ \mathbf{u} &= \frac{\mathbf{p}}{\mathbf{p}^{\mathrm{T}}\mathbf{v}} - \frac{\mathbf{G}^k\mathbf{v}}{\mathbf{v}^{\mathrm{T}}\mathbf{G}^k\mathbf{v}} \end{aligned}$$

$\mathbf{G}$ is initialized to the identity.

The gradient $\mathbf{g}$ is calculated by the standard backpropagation algorithm (see, for example, [12]), as follows. From Equation 4.7, the elements of the gradient vector are given by

$$\frac{\partial E}{\partial w_{ij}} = \sum_p \frac{\partial E_p}{\partial w_{ij}} \tag{4.15}$$

To simplify notation, we can calculate $\frac{\partial E_p}{\partial w_{ij}}$ and drop the subscript p from all of the network quantities. n_i is the net input to unit i, a_i is the activation of unit i, and w_{ij} is the weight from unit j to unit i.

$$\begin{aligned} \frac{\partial E_p}{\partial w_{ij}} &= \sum_m e_m \frac{\partial o_m}{\partial w_{ij}} \\ &= \sum_m e_m \frac{\partial o_m}{\partial n_i}\frac{\partial n_i}{\partial w_{ij}} \\ &= a_j \sum_m e_m \frac{\partial o_m}{\partial n_i} \\ &= \delta_i a_j \end{aligned} \tag{4.16}$$

$$\delta_i =: \sum_m e_m \frac{\partial o_m}{\partial n_i} \tag{4.17}$$

δ_i is computed for each layer recursively. For units in the output layer:

$$\begin{aligned} \delta_i &= \sum_m e_m \frac{\partial o_m}{\partial n_i} \\ &= \sum_m e_m \gamma'(n_i) \end{aligned} \tag{4.18}$$

where $\gamma'(n_i)$ is the first derivative of the activation function $\gamma(n_i)$ with respect to its argument. If $\gamma(n_i) = \frac{1}{1+e^{-\beta n_i}}$, the standard sigmoidal function, then

$$\begin{aligned} \delta_i &= \sum_m e_m \beta \gamma(n_i)(1-\gamma(n_i)) \\ &= \beta a_i(1-a_i) \sum_m e_m \end{aligned} \tag{4.19}$$

For units i in the hidden layers, δ_i is computed from the δ_ks of all units k in the next layer toward the output:

$$\begin{aligned} \delta_i &= \sum_m e_m \frac{\partial o_m}{\partial n_i} \\ &= \sum_m e_m \sum_k \frac{\partial o_m}{\partial n_k} \frac{\partial n_k}{\partial n_i} \\ &= \sum_k \delta_k \frac{\partial n_k}{\partial n_i} \\ &= \sum_k \delta_k w_{ki} \frac{\partial a_i}{\partial n_i} \\ &= \sum_k \delta_k w_{ki} \gamma'(n_i) \end{aligned} \tag{4.20}$$

Again, with the sigmoidal activation function:

$$\delta_i = \beta a_i(1-a_i) \sum_k \delta_k w_{ki} \tag{4.21}$$

The coefficient α in Equation (4.13) is determined by performing a line search for a minimum along the direction $-\mathbf{G}^k(\mathbf{g}^k)^{\mathrm{T}}$. One method for performing such a search is the quadratic fit method [60]. This method has the advantage of not requiring calculation of the gradient of the function $E(\alpha)$ to be minimized, but only calculation of the function itself. Three initial points, α_1, α_2, and α_3, corresponding to different values of α along the search line, are chosen such that α_1 and α_3 bracket a minimum ($E(\alpha_1)$, $E(\alpha_3) > E(\alpha_2)$). The corresponding values of the function, $E_i = E(\alpha_i)$, are calculated. Then, approximating the error surface along the search line as a quadratic function through the three points, one obtains:

$$\tilde{E}(\alpha) = \sum_{i=1}^{3} E_i \frac{\prod_{j\neq i}(\alpha - \alpha_j)}{\prod_{j\neq i}(\alpha_i - \alpha_j)} \tag{4.22}$$

A guess at the location of the minimum of E along this line is obtained by minimizing this quadratic function:

$$\alpha_4 = \frac{(\alpha_2^2 - \alpha_3^2)E_1 + (\alpha_3^2 - \alpha_1^2)E_2 + (\alpha_1^2 - \alpha_2^2)E_3}{2[(\alpha_2 - \alpha_3)E_1 + (\alpha_3 - \alpha_1)E_2 + (\alpha_1 - \alpha_2)E_3]} \tag{4.23}$$

This new point α_4 can then be used together with two of α_1, α_2, and α_3 to generate an improved quadratic. If $E_4 < E_2$, α_4 becomes the new middle point, and the new triplet is $(\alpha_1, \alpha_4, \alpha_2)$ or $(\alpha_2, \alpha_4, \alpha_3)$, depending on whether $\alpha_4 < \alpha_2$ or $\alpha_4 > \alpha_2$. Otherwise, if $E_4 > E_2$, α_4 becomes a new end point, and the new triplet is $(\alpha_4, \alpha_2, \alpha_3)$, or $(\alpha_1, \alpha_2, \alpha_4)$, depending again on whether $\alpha_4 < \alpha_2$ or $\alpha_4 > \alpha_2$.

Any of a variety of stopping criteria can be used to end the search algorithm; a simple one, used in the network training here, is the percentage test. With this method, the algorithm is halted when the coefficient α has been determined within a fixed percentage of its true value. At each step in the algorithm, the value of α which yields the minimum of E, α^*, lies between the two outer points, α_1 and α_3, and the estimate of α^* is $\hat{\alpha} = \alpha_2$. A simple bound on the percentage error is therefore given by

$$\frac{|\hat{\alpha} - \alpha^*|}{\alpha^*} < \frac{\max(\alpha_3 - \alpha_2, \alpha_2 - \alpha_1)}{\alpha_1} \tag{4.24}$$

The percentage cutoff used for stopping the line search in the neural network training was .01. Details about the convergence properties of this line search algorithm can be found in [60]. Note that, as discussed above, the quasi-Newton algorithm can be used for batch learning only; to perform on-line learning on a pattern basis, a different method, such as gradient descent, must be used.

To generate the initial triplet α_1, α_2, α_3 for the quadratic fit method, a coarse search along the line is performed as follows. α_1 is initialized to zero. Then, α_2 is incremented from $\alpha_2 = \alpha_1$ until $E_2 < E_1$. If such a point cannot be found within a fixed number of steps (chosen as 10 here), α_2 is reset to zero and the increment size (here initialized as .1) is reduced by a factor of ten. Once α_2 is fixed, α_3 is incremented from $\alpha_3 = \alpha_2$ until $E_3 > E_2$. Again, if such a point cannot be found within a certain number of steps (for this second stage, 100 was chosen), α_3 is reset to α_2 and the increment size is reduced by a factor of ten. If, while attempting to initialize either α_2 or α_3, the size of the increment drops below a cutoff (here, .00001), the triplet generation algorithm returns an error. In this case, the quasi-Newton algorithm assumes that either a local minimum has been reached or, more likely, the estimate of the inverse Hessian, $\mathbf{G}$, is poor, yielding a bad choice of search direction. The latter is likely to happen near the beginning of the quasi-Newton descent. When an error is returned, the descent algorithm takes a small step ($\alpha = .000001$) along a new direction (no longer equal to $-\mathbf{G}\mathbf{g}^{\mathbf{T}}$) whose components are randomly distributed between -.5 and .5.

Data Set Generation

To train the control networks, a large data set was generated by simulating the effects of randomly chosen control vectors lying within the ranges shown in Table 4.1. The initial position of the controlled DOF was selected randomly from within its allowed range; all other degrees of freedom were fixed at the zero angle (except for degrees of freedom 4 and 5 in the shoulders, which were set at -.1 to avoid the Euler singularities). The data sets for DOF 2 are shown in Appendix B. The data, both input and output, were then scaled, using a piecewise affine transformation described below, to lie in $[0, 1]$, again using the ranges shown in Table 4.1. The output $\mathbf{y}$ data ranges were chosen to delineate the values allowed for $\mathbf{y_d}$ selection. For the change in position $\Delta\theta$, this scaling range is exactly twice the joint range shown in Table 3.3, though the maximum values were not always achievable by the allowed controls. Data samples in which the simulation ran up against the joint limits or in which the change in velocity $\Delta\dot{\theta}$ was outside the preselected range of $\Delta\dot{\theta}_d$ were discarded. The minimum and maximum T values as well as the control value ranges were selected to produce torque profiles compatible with the biological data presented in [34]. The pulse height ranges, of course, are scaled when they pass through the inertial compensator.

In order to indicate direction, the scaled range of the first control parameter (either $\mathrm{pw_1}$ or $\mathrm{ph_1}$) is divided into two parts. Values in $[0, .5)$ correspond to negative torque (or desired acceleration) pulses, while values in $(.5, 1]$ indicate positive ones (a value of exactly .5 is indeterminate). The second pulse is always interpreted to be in the opposite direction from the first pulse. Thus, if p_{min} is the minimum unscaled value of the first control parameter p, and p_{max} is the maximum value, then the scaled value p_s of p is given by:

$$p_s = .5 + .5\mathrm{sgn}(p)\frac{(|p| - p_{\text{min}})}{(p_{\text{max}} - p_{\text{min}})}, \tag{4.25}$$

where $\mathrm{sgn}(p) = 1$ for $p > 0$ and $\mathrm{sgn}(p) = -1$ otherwise. The second control parameter is scaled as:

$$p_s = \frac{(|p| - p_{\text{min}})}{(p_{\text{max}} - p_{\text{min}})} \tag{4.26}$$

The final control parameter, as well as $\dot{\theta}_0$ and the three the resultant movement parameters, are scaled as:

$$p_s = \frac{(p - p_{\min})}{(p_{\max} - p_{\min})} \tag{4.27}$$

Because the pulse width values are larger for PWM than for PHM, the PWM regime produces movements with larger T, as desired. The cutoff value of T_d which determines whether a movement will be executed by PWM or PHM was derived from the generated data sets and fixed at .29735 seconds (a scaled value of .421). In simulation, T is measured as the time when the feedforward torque has decayed to 5% of its second peak value. (For the single-DOF controller data generation and training, this criterion was checked every .001 seconds; for the full dive simulations described in Section 4.2, large apparent forces during the dive required that this criterion be checked at every integration step.) The pulse height ranges in both control schemes were chosen to make data generation easy without much compromise in the range of ys produced. The largest pulse heights in the PWM control mode were eliminated because they caused the simulation to run into the joint limits extremely frequently. Similarly, the smallest pulse heights in the PHM control mode were eliminated because a combination of a large first pulse with a small second pulse, or a small first pulse with a large second pulse, often resulted in the simulation running into the joint limits or exceeding the preselected $\Delta\dot{\theta}$ range. The largest pulse heights were needed in this mode to achieve the desired range of output behavior.

1000 data points were generated for each DOF and mode (PWM or PHM) on the right side of the body (the even numbered degrees of freedom). With fewer data points, the trained nets did not generalize well, but more points did not improve generalization and slowed down training. A second, disjoint set of 100 data points was also generated for each DOF and mode for use in testing the network generalization. In generating the 1000-point data sets, no points whose scaled first control value was within .001 of the discontinuity (that is, in $(.499, .501)$) were permitted; this helped the networks train more efficiently, eliminating constraints on their output near the discontinuity. For the 100-point test data set, points whose first control value was within .01 of the discontinuity were eliminated.

Training Results

After the data sets were generated, the networks for each mode of each even DOF were trained using the quasi-Newton method discussed above. First, the forward network was

	Parameter	Range
	$\Delta\theta$	$[\theta_{\min} - \theta_{\max}, \theta_{\max} - \theta_{\min}]$
	$\Delta\dot{\theta}$	$[-6.9744, 6.9744]$
	T	$[.15, .5]$
	$\dot{\theta}_0$	$[-3.4872, 3.4872]$
PWM	pw_1	$[-.18, -.11]$, $[.11, .18]$
	pw_2	$[-.18, -.11]$, $[.11, .18]$
	ph	$[12, 320]$
PHM	ph_1	$[-400, -60]$, $[60, 400]$
	ph_2	$[-400, -60]$, $[60, 400]$
	pw	$[.05, .11]$

Table 4.1: Scaling ranges for network inputs and outputs. Each parameter value p was scaled such that $p_s = \frac{p - p_{\min}}{p_{\max} - p_{\min}}$, except for the first and second control parameters, as discussed in the text. Angles are in radians, angular velocities are in radians per second, times and pulse widths are in seconds, and pulse heights are in radians per second squared, the units of the desired acceleration signal. The values $\theta_{\min}$ and $\theta_{\max}$ are those in Table 3.3.

trained until the total squared error, measured as $\sum_p \sum_m (o_{p,m} - t_{p,m})^2 = 2\sum_p E_p$, on the 1000 scaled training data points was less than .08. Then the corresponding inverse network was loosely trained using the direct method, using the error on the controls, until its total squared error was less than .8. After this preliminary training, the inverse network was trained using the indirect method, as described above, until its total squared error $2\sum_p E_p$ was less than .08. The initial direct training was found to help the indirect training avoid local minima.

The networks were considered adequately trained if three criteria were met:

1. The total squared error $2\sum_p E_p$ on the disjoint test set of 100 data points was less than .025 both for the forward net and for the inverse and forward nets combined (error measured as $\sum_p ||\hat{\mathbf{y}}_\mathbf{p} - \mathbf{y}_{\mathbf{p},\mathbf{d}}||^2$ at the output of the forward net),

2. The total squared output error produced by simulating the controls generated by the inverse net for the 100 test data points was less than .025, and

3. The total squared output error produced by simulating the controls generated by the inverse net for a newly-generated set of 100 test points (see below) was less than .05.

If the networks failed any of these tests, or if the indirect training of the inverse network converged slowly or converged with an asymptotic error greater than .08, the training was refined by either further training the forward net to a smaller error cutoff (in decrements of .02), further training the inverse net in the indirect method to a smaller error cutoff (also in decrements of .02), or, in cases of slow convergence or local minima in the indirect training, further training the inverse net in the direct method to a smaller error cutoff (in decrements of .2). If these refinements failed to produce adequately trained networks, the weights of one or both of the networks were randomized and the process restarted.

These tests of the learned weights were done with all DOF angles randomized to lie between 10% and 90% of their range. The starting position θ_0 of the controlled DOF was adjusted with a heuristic algorithm to ensure that the simulation would not come up against the joint limits. The heuristic made use of the desired angle change, $\Delta\theta_d$, and the proposed control pulse heights and widths to determine the likely final position of the DOF and likely peaks in the angle trajectory that would violate the joint limits.

Table 4.2 shows the training and testing results for the single-DOF networks, and Appendix C provides training error convergence plots for the DOF 0, PHM networks, more detailed plots of final training errors for the DOF 2, PWM mode network, and sample movements produced by a few of the trained single-DOF controllers (see Table 3.3 for the numbering of the degrees of freedom; DOF 0 is the right hip, and DOF 2 is one of the Euler angles in the right shoulder). Training the forward network generally took on the order of one or two thousand iterations; training the inverse network generally took a few hundred iterations of direct training followed by a few thousand iterations of indirect training. The number of iterations required for indirect training to the cutoff error varied widely among the different networks.

The third test of the trained networks involves newly generated data points, as mentioned above. As not all possible inputs $\mathbf{y}_d$ are achievable with the allowed controls, it is useful to use the forward net as a test to determine whether a particular $\mathbf{y}_d$ is feasible. Since the inverse net uses sigmoidal outputs, any output from this net lies in $(0,1)$ and is thus a valid control, though outputs for which the first control value lies within .01 of the discontinuity, that is, within $(.49, .51)$, may be unreliable. If the control generated by the inverse net is passed through the forward net, the resulting output $\hat{\mathbf{y}}$ can be compared with $\mathbf{y}_\mathbf{d}$. If the squared error $||\hat{\mathbf{y}} - \mathbf{y}_\mathbf{d}||^2$ is greater than a certain cutoff, chosen here as .0015, the proposed $\mathbf{y}_\mathbf{d}$ is rejected as being outside the range of the controls. Further, the $\mathbf{y}_\mathbf{d}$ is also rejected if the heuristic described above for placing the initial DOF angle, θ_0, fails to find a θ_0 that is not likely to result in a joint limit violation during the movement. For the

		Training		Test Data	New Data
Joint	Mode	Cutoffs	Test Data	(Simulated)	(Simulated)
0	PWM	.06, .08	.0065, .0103	.0137	.0332, 343 rejected
2	PWM	.08, .08	.0077, .0079	.0180	.0250, 292 rejected
4	PWM	.08, .08	.0099, .0089	.0180	.0294, 292 rejected
6	PWM	.08, .08	.0137, .0096	.0178	.0304, 258 rejected
8	PWM	.06, .08	.0068, .0058	.0113	.0260, 282 rejected
0	PHM	.06, .08	.0188, .0063	.0087	.0225, 198 rejected
2	PHM	.06, .08	.0079, .0095	.0140	.0282, 179 rejected
4	PHM	.04, .08	.0051, .0080	.0093	.0199, 131 rejected
6	PHM	.04, .08	.0165, .0106	.0106	.0252, 308 rejected
8	PHM	.06, .06	.0045, .0180	.0191	.0294, 134 rejected

Table 4.2: Training and testing data for the single-DOF controllers. The training cutoffs are the total squared errors $\sum_p \sum_m (o_{p,m} - t_{p,m})^2 = 2\sum_p E_p$ for the forward network and for the combined inverse and forward networks (after indirect training, with error measured as $\sum_p ||\hat{\mathbf{y}}_\mathbf{p} - \mathbf{y}_{\mathbf{p},\mathbf{d}}||^2$) on the 1000 training data points. The next column, Test Data, gives the total squared error for the forward network and for the two combined networks tested on the 100-point test data set. The following column gives the error $\sum_p ||\hat{\mathbf{y}}_\mathbf{p} - \mathbf{y}_{\mathbf{p},\mathbf{d}}||^2$ in the output when the controls generated by the inverse network for the test data points are simulated. The final column reports the error on 100 freshly generated data points which were tested for feasibility by being passed through the forward network (see text). The number of rejected $\mathbf{y}_\mathbf{d}$s is also indicated. All errors are measured on scaled data.

third test of each trained network, randomly generated $\mathbf{y}_d$s were tested for feasibility in this manner; the controls generated by the network for 100 feasible points were then simulated. The resulting errors are shown in the final column of Table 4.2, along with the number of rejected $\mathbf{y}_d$s, and in the bottom plot of Figure C.4 .

Training was facilitated by noting that the corresponding networks for the left and right sides of the body approximate the same functions, except for slight variations in the arm adduction/abduction DOF due to the reversed influence of gravity, so only one set of weights needs to be trained. The influence of gravity on the function modeled by the trained networks is only slight because the feedback around the desired acceleration profile largely compensates for gravity. Furthermore, any networks for which the $\mathbf{y}$ scalings are the same approximate the same functions, except, again, for differing influences of gravity. Training time was reduced for several of the degrees of freedom by initializing their training with weights from other degrees of freedom that had already been trained: in both PWM and PHM modes, the weights for degrees of freedom 2 and 8 were initialized with the trained weights from DOF 0. These networks needed some further training to achieve the error cutoffs because the behavior of the networks near the discontinuity in the first control parameter varies widely and is highly dependent on the particular data set used to train the weights. Thus, when transferring weights from one network to another, first control parameters in (.49,.51) were sometimes generated inaccurately, resulting in larger training errors. This variability and inaccuracy near the control discontinuity is the motivation for disallowing controls in $(.49, .51)$, both in the network test described above and in the use of the trained networks in the complete control system (see Section 4.2.1). The weights for the odd degrees of freedom (the left side of the body) were copied from the corresponding even degrees of freedom with no retraining.

4.2 Coordinating Controller

4.2.1 Design

A closeup of the coordinating controller architecture is shown in Figure 4.4. The hierarchical, behavioral design of this controller is motivated by the biological idea of multi-joint synergies as subprograms in complex skills discussed in Section 2.5. Each dive is built out of a series of parametrized multi-joint behavioral synergies. These synergies serve to coordi-

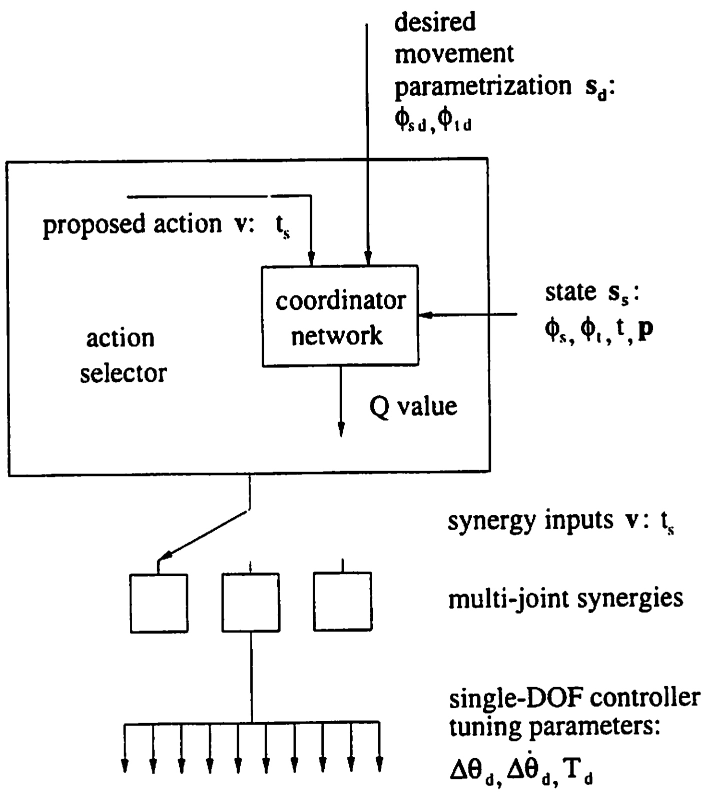

desired movement parametrization s_d: ϕ_{sd}, ϕ_{td}
proposed action v: t_s
coordinator network
action selector
state s_s: $\phi_s, \phi_t, t, \mathbf{p}$
Q value
synergy inputs v: t_s
multi-joint synergies
single-DOF controller tuning parameters: $\Delta\theta_d, \Delta\dot{\theta}_d, T_d$

Figure 4.4: Design of the coordinating controller.

nate the single-DOF controllers to produce stereotypical multi-joint behavior patterns such as "go into the pike position" or "execute the throw maneuver." The coordinating controller itself takes as input a vector $\mathbf{s} = [\mathbf{s_d}, \mathbf{s_s}]$ containing the desired somersault and twist angles for the dive, $\mathbf{s_d} = [\phi_{sd}, \phi_{td}]$, plus sensory information $\mathbf{s_s} = [\phi_s, \phi_t, t, \mathbf{p}]$ describing the current somersault and twist angles, the current time, and the current body posture. The body position $\mathbf{p}$ is a binary vector with each value corresponding to the overall position achieved by one of the multi-joint synergies. This simplification of the diver's state representation is crucial to reducing the number of network inputs, at the cost of having only approximate state information. The controller outputs tuning parameters $\mathbf{v}$ for the multi-joint synergies. In general, these tuning parameters could include such things as the desired tightness of a pike or the speed with which the multi-joint synergy should be executed, but in this implementation, the tuning parameter vector $\mathbf{v}$ contains only t_s, the time to wait before initiating the behavior; all other parameters in the multi-joint synergies are fixed. Only one synergy can be active at a time; the controller switches among them, behaving like a a timed state machine, with a different state machine (ordering of the multi-joint synergies) for each type of dive.

Each multi-joint synergy takes as input the vector $\mathbf{v} = [t_s]$ and outputs the control specifications $\mathbf{y_d}$ for all of the single-DOF controllers. The three behavioral synergies used for the diver are "throw," "pike," and "layout;" each specifies a desired position for each degree of freedom, as shown in Table 4.3. The desired final velocity is always zero. For a given synergy, the desired movement times for all degrees of freedom are the same, as shown in Table 4.3. Thus, all degrees of freedom should finish their movements simultaneously in a multi-joint action. The θ_d and T_d values were chosen to be similar to values used by human divers as well as feasible for the single-DOF controllers to execute. With the T cutoff of slightly less than .3 seconds (see Section 4.1.2), the throw synergy is produced by the PHM mode networks, while the others use PWM mode. These values are used, together with the DOF angles and velocities at the start of the behavior, to compute the inputs $\mathbf{y_d} = [\Delta\theta_d, \Delta\dot{\theta}_d, T_d]$ to the single-DOF controllers. The high apparent torques present during the dive make it necessary to stiffen the single-DOF PD controllers acting to achieve the desired joint kinematics (see Section 4.1.1) to $k_p = 3750$, $k_d = 3750$. If the current position and velocity of a DOF is close enough to the desired values (within .175 radians and .75 radians/second, respectively), then the single-DOF controller is not invoked, but a new PD controller ($k_p = 3750$, $k_d = 750$) centered at the desired position and zero velocity

	θ_d										T_d
	0	1	2	3	4	5	6	7	8	9	
Throw	0	0	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	$-\frac{\pi}{8}$	$-\frac{7\pi}{8}$	$\frac{\pi}{2}$	$-\frac{\pi}{2}$	$-\frac{\pi}{2}$	$-\frac{\pi}{2}$	.26
Pike	$-\frac{4\pi}{5}$	$-\frac{4\pi}{5}$	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	$-\frac{\pi}{2}$	$-\frac{\pi}{2}$	0	0	0	0	.30 / .38
Layout	0	0	0	0	$.1-\pi$	$.1-\pi$	0	0	0	0	.32

Table 4.3: Specification of the three behaviors used in the coordinating controller. Each row indicates the desired angle for each degree of freedom (in radians) and the desired movement time (in seconds) for one of the behaviors. The desired final velocity is always zero. The pike behavior had two different desired durations depending on the dive being performed; for the jackknife and the one-and-one-half-somersault pike, its T_d was .38 seconds, and for the one-and-one-half-somersault, full-twist dive, T_d for the pike was .3 seconds. Degrees of freedom 0 and 1 correspond to the right and left hip, 2, 4, and 6 are the three Euler angles in the right shoulder, 3, 5, and 7 are the angles of the left shoulder, and 8 and 9 are the right and left elbow. In the layout synergy, the angles for degrees of freedom 4 and 5 were chosen to be slightly greater than $-\pi$ to avoid the Euler singularities.

is activated. This same PD controller is active between multi-joint synergies, with set points determined by the DOF angles at the end of the previous synergy. The $\mathbf{y_d}$ values are passed through each DOF's inverse network to select a control vector $\mathbf{u}$, which is then passed through the DOF's forward network to obtain an estimated outcome $\hat{\mathbf{y}}$. As for the network test discussed in Section 4.1.2, if the error $||\hat{\mathbf{y}} - \mathbf{y_d}||^2 > .0015$ the $\mathbf{y_d}$ is considered unreachable. A dive being executed is aborted if such a bad request is made. A multi-joint behavior is considered complete when each of the single-DOF controllers it has activated have finished their movements.

The implementation of the control system described in this chapter learns to perform three dives: a jackknife, a one-and-one-half-somersault pike, and a one-and-one-half-somersault, full-twist dive. For the jackknife, the diver must execute a pike and enter the water after a somersaulting rotation of π radians (so $\phi_{sd} = \pi$, $\phi_{td} = 0$). The pike position involves bringing the legs up and the arms out to decrease the moment of inertia about the somersaulting axis. For the one-and-one-half-somersault pike, the diver must execute a pike and enter the water with an overall somersault angle of 3π radians ($\phi_{sd} = 3\pi$, $\phi_{td} = 0$). Finally, for the twisting dive, the diver must first execute a throwing maneuver with the arms to tilt the body's rotation axis (see [27]), perform a full 2π twist, then go into the pike position, coming out at the end of the dive with 3π radians of somersault rotation ($\phi_{sd} = 3\pi$, $\phi_{td} = 2\pi$).

The dive simulation is started from a set of initial conditions that is fixed for each dive. The body somersault angle ϕ_s is initialized to .175 radians, based on an inspection of Olympic diving footage (described in more detail in Section 4.2.3), and the twist angle ϕ_t is initialized to 0, for all dives. The initial joint angles for all dive types are shown in Table 4.4. The hip angles (degrees of freedom 0 and 1) are set to -.175 so that the legs start out vertical. The initial angles for the one-and-one-half-somersault, full-twist dive correspond to a position with the arms straight out away from the sides of the body. The initial velocities of all degrees of freedom are zero, except for the twisting dive, in which several degrees of freedom start with ± 3.4872 radians/second, the maximal velocities allowable; degrees of freedom 0 and 1 start with a velocity of -3.4872 radians/second and degrees of freedom 4 and 5 start with a velocities of 3.4872 and -3.4872 radians/second, respectively. The initial velocities for degrees of freedom 4 and 5 help the diver achieve a higher twist velocity with the throw maneuver. The initial velocities for degrees of freedom 0 and 1 are required for the single-DOF controllers to be able to produce the throw movement; they also help make the platform takeoff look slightly more natural. The initial vertical velocity of the center of mass of the body is 3 m/s; this value was chosen to be consistent with the initial velocities used by human divers. The vertical velocity, along with the initial height of the center of mass, determines the time available for the execution of the dive before the diver hits the water. The initial forward velocity of the center of mass of the body is .75 m/s, enough so that the diver will not hit the board. The initial vertical position of the center of mass varied among the dives since the initial joint positions varied (see Table 4.4). The vertical position was chosen so that the diver's feet would start at a fixed position (on the diving platform). The simulation time for each dive was determined based on the vertical distance the center of mass must travel. This distance was equal to the initial vertical position of the center of mass plus 10 meters (the height of the diving platform) minus the approximate distance from the center of mass to the tips of the fingers in the layout position (a length equal to 1.059 meters). This distance through which the center of mass must fall, together with the initial vertical velocity, yielded dive times of 1.766 seconds for 9.989 meters for the jackknife and one-and-one-half-somersault pike dives, and 1.764 seconds for 9.971 meters for the twisting dive. The somersaulting angular momentum components for the dives were set to 16 kg·m^2 for the jackknife, 47 kg·m^2 for the one-and-one-half-somersault pike, and 65 kg·m^2 for the twisting dive. The twisting momentum component was zero for both the jackknife and the one-and-one-half-somersault pike, but was set to 5 kg·m^2 for the twisting

	θ_0									
	0	1	2	3	4	5	6	7	8	9
jackknife	-.175	-.175	0	0	$.1-\pi$	$.1-\pi$	0	0	0	0
$1\frac{1}{2}$ som. pike	-.175	-.175	0	0	$.1-\pi$	$.1-\pi$	0	0	0	0
$1\frac{1}{2}$ som. full twist	-.175	-.175	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	$-\frac{\pi}{2}$	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	$-\frac{\pi}{2}$	0	0

Table 4.4: Initial angles for each DOF for each dive type. The hip angles (degrees of freedom 0 and 1) are set to -.175 so the legs start out vertical (see text). Degrees of freedom 0 and 1 correspond to the right and left hip, 2, 4, and 6 are the three Euler angles in the right shoulder, 3, 5, and 7 are the angles of the left shoulder, and 8 and 9 are the right and left elbow. For the jackknife and the one-and-one-half-somersault pike, degrees of freedom 4 and 5 are set to slightly larger than $-\pi$ to avoid the Euler angle singularities. All angles are in radians.

dive. The desired end joint configuration is the same for all dives, with the angles of all degrees of freedom being zero except for degrees of freedom 4 and 5, which are at $-\pi + .1$. The desired final velocity of all joints is zero.

The coordinating controller is trained in two stages inspired by the stages of motor learning discussed in Chapters 1 and 2. In the cognitive, "learning by watching" stage, a diving student learns about the task goal and the behavioral structure of the task from watching other divers and from instructor input. During this stage, the student also learns an initial estimate of the coordination parameters $\mathbf{v}$. For the coordinating controller, the learning by watching stage consists of defining the dive's cost function (or functions; see Sections 4.2.2–4.2.3), breaking up the dive into multi-joint synergies, defining the order in which these synergies are to be activated, and learning a rough estimate of the coordination timing t_s required to produce the dive. In the second, "learning by doing," training stage, which corresponds to the associative and autonomous learning stages in human skill acquisition, the controller practices the dive to refine the coordination parameters.

The internal structure of the coordinating controller is intimately tied to the algorithm chosen for implementing the "learning by doing" parameter refinement phase. This algorithm, called Q-learning, is described in the next section; its implementation in the controller network is discussed in Section 4.2.3.

4.2.2 Reinforcement Learning Theory

The diving problem, whose solution requires a correctly-timed sequence of behaviors, is well addressed in the framework of reinforcement learning. Reinforcement learning theory is an adaptation of dynamic programming algorithms to the type of task faced by the coordinating controller: the controlling agent must select an action based on the current state of the system, and must, with little or no prior knowledge about the structure of the problem or the probabilities of various outcomes of the control action, learn which actions will lead to the largest rewards or smallest costs. The costs simply assign some value to a state or transition rather than providing information about the correct choice, as in supervised learning, and might be given only at the end of a complete movement iteration, as in the diving example. The reinforcement learning algorithms presented in this section are techniques for addressing the temporal credit assignment problem, the question of how to assign credit among a series of control choices all contributing to the value of a delayed outcome evaluation. Most of the following discussion is based on the presentation in [11].

The framework in which dynamic programming algorithms are applied is that of a Markov decision process (MDP). An MDP is a generalization of a Markov process (a set of states $\mathcal{S}$ with transition probabilities $p(\mathrm{s_i}, \mathrm{s_j})$, $\mathrm{s_i}, \mathrm{s_j} \in \mathcal{S}$) in which the state transition probabilities are dependent on the choice of a control action $\mathbf{v} \in \mathcal{V}(\mathrm{s_i})$; that is, $p = p(\mathrm{s_i}, \mathbf{v}, \mathrm{s_j})$, or, to simplify notation, $p_{i,j}(\mathbf{v})$. In the dynamic programming framework, the state space is discrete; for the diver problem, though, we have $\mathbf{s} = [\mathbf{s_d}, \mathbf{s_s}] = [\phi_{sd}, \phi_{td}, \phi_s, \phi_t, t, \mathbf{p}]$ (a combination of continuous and discrete components) and $\mathbf{v} = [t_s]$ (a continuous variable), as described in Section 4.2.1. Each transition $\mathrm{s_i} \to \mathrm{s_j}$ with action $\mathbf{v}$ is also assigned a cost $c(\mathrm{s_i}, \mathbf{v}, \mathrm{s_j})$, or, again to simplify notation, $c_{i,j}(\mathbf{v})$. The basic dynamic programming algorithms solve for $J^*\colon \mathcal{S} \to \mathbb{R}$, the "value" or optimal cost-to-go function, which is the minimal expected cost that can be accrued over a trajectory of states $\mathrm{s_j}$ starting in each state $\mathrm{s_i}$:

$$J^*(\mathrm{s_i}) = \min_{\mu} \mathrm{E}\left[\sum_{j=i}^{\infty} \gamma^{j-i} c_{j,j+1}(\mathbf{v_j})\right] \tag{4.28}$$

where $\mu\colon \mathbf{s} \mapsto \mathbf{v}, \mathbf{s} \in \mathcal{S}, \mathbf{v} \in \mathcal{V}$, the function defining the choice of action in each state, is called the policy, and γ, $0 < \gamma \leq 1$, is a discounting factor. Equation (4.28) is written for an infinite-horizon problem, but finite-horizon problems (possibly with state-dependent termination costs) can be considered in the same framework. In the dynamic programming framework, J^* can be written as a vector, because the state space is discrete, and can therefore be stored in a lookup table. J^* must satisfy Bellman's equation

$$\begin{aligned} J^*(\mathrm{s_i}) &= \min_{\mathrm{v}\in\mathcal{V}(\mathrm{s_i})} \mathrm{E}[c_{i,j}(\mathrm{v}) + \gamma J^*(\mathrm{s_j})] \\ &= \min_{\mathrm{v}\in\mathcal{V}(\mathrm{s_i})} \sum_{s_j} p_{i,j}(\mathrm{v})\,(c_{i,j}(\mathrm{v}) + \gamma J^*(\mathrm{s_j})) \end{aligned} \tag{4.29}$$

This system of equations can be solved either directly or using the dynamic programming update called value iteration:

$$J^{k+1}(\mathrm{s_i}) = \min_{\mathrm{v}\in\mathcal{V}(\mathrm{s_i})} \sum_{s_j} p_{i,j}(\mathrm{v})(c_{i,j}(\mathrm{v}) + \gamma J^k(\mathrm{s_j})), \tag{4.30}$$

Using this algorithm, $J \to J^*$ as $k \to \infty$; J^* is the unique fixed point of the Bellman equation (4.29). The algorithm converges if the $J(\mathrm{s_i})$ are updated synchronously (updates are done for all $\mathrm{s_i}$ simultaneously) or asynchronously (updates are done for one $\mathrm{s_i}$ at a time), as long as $J(\mathrm{s_i})$ is updated infinitely often for each $\mathrm{s_i}$. Once J^* is known, the optimal choice of action in each state (the optimal policy), $\mu(\mathrm{s_i})$, is obtained by:

$$\mu(\mathrm{s_i}) = \arg\min_{\mathrm{v}\in\mathcal{V}(\mathrm{s_i})} \sum_{s_j} p_{i,j}(\mathrm{v})(c_{i,j}(\mathrm{v}) + \gamma J^*(\mathrm{s_j})) \tag{4.31}$$

Policy iteration is a method for determining the best policy more directly. In this technique, a policy μ_k: $\mathrm{s} \mapsto \mathrm{v}, \mathrm{s} \in \mathcal{S}, \mathrm{v} \in \mathcal{V}$ is evaluated by solving for J^{μ_k},

$$J^{\mu_k}(\mathrm{s_i}) = \sum_{s_j} p_{i,j}(\mu(\mathrm{s_i}))(c_{i,j}(\mu(\mathrm{s_i})) + \gamma J^{\mu_k}(\mathrm{s_j})), \tag{4.32}$$

either by solving this set of equations directly or by using value iteration. Then a policy improvement stage is carried out, which selects the optimal action in each state based on J^{μ_k}:

$$\mu^{k+1}(\mathrm{s_i}) = \arg\min_{\mathrm{v}\in\mathcal{V}(\mathrm{s_i})} \sum_{s_j} p_{i,j}(\mathrm{v})(c_{i,j}(\mathrm{v}) + \gamma J^{\mu_k}(\mathrm{s_j})) \tag{4.33}$$

The value $J^{\mu_{k+1}}$ of the new policy μ^{k+1} can then be evaluated, and the process repeated. This algorithm will converge to an optimal policy μ^*.

These dynamic programming algorithms all require complete knowledge of the system's transition probabilities and costs. If an explicit system model is not available, as is typical in reinforcement learning problems, but simulation is possible, one approach is to simulate many trajectories and use them to estimate the transition probabilities and costs. An alternative, iterative approach to policy evaluation in the absence of an explicit model is provided by the temporal difference family of methods. These methods, called TD(λ), use the trajectory-based update rule

$$\begin{aligned} J^{k+1}(\mathrm{s_i}) &= J^k(\mathrm{s_i}) + \alpha \sum_{j=i}^{\infty} (\gamma\lambda)^{j-i} d_j \\ d_j &= c_{j,j+1}(\mu(\mathrm{s_j})) + \gamma J^k(\mathrm{s_{j+1}}) - J^k(\mathrm{s_j}) \end{aligned} \tag{4.34}$$

where $0 \leq \lambda \leq 1$ and α is a step size parameter. Note that this algorithm applies for a fixed policy μ; $\mathrm{s_{j+1}}$ is the state reached in a simulated trajectory after $\mu(\mathrm{s_j})$ is applied. d_j is called the temporal difference. d_j is an estimate of the error of the cost-to-go $J(\mathrm{s_j})$ based on the simulated transition from $\mathrm{s_j}$ to $\mathrm{s_{j+1}}$; the transition accumulated a cost of $c_{j,j+1}(\mu(\mathrm{s_j}))$, and after the transition, the current estimate of the discounted cost-to-go from the new state is $\gamma J(\mathrm{s_{j+1}})$. If the system is stochastic, d_j is a random variable. The TD(λ) algorithm replaces knowledge of the transition probabilities with stochastic simulation of the trajectories, so that the probability of a particular $\mathrm{s_{j+1}}$ and the d_j it produces will reflect the system transition probabilities. The updates in Equation (4.34) can be done in an offline manner after the completion of an entire (finite) trajectory or in an online manner, one term at a time, with updates after each state transition. In either case, TD(λ) will converge to J^μ with probability one, as long as each state is visited infinitely often (which requires infinite restarts for finite-horizon problems) and the step size α is decreased at an appropriate rate.

Note that the extreme case of TD(λ) in which $\lambda = 1$, TD(1), is equivalent to using an incremental update based on the actual cost of a simulated trajectory. Rewriting Equation (4.34) we obtain:

$$J^{k+1}(\mathrm{s_i}) = (1-\alpha) J^k(\mathrm{s_i}) + \alpha \sum_{j=i}^{\infty} \gamma^{j-i} c_{j,j+1}(\mu(\mathrm{s_j})) \tag{4.35}$$

Versions for which $\lambda < 1$ place less emphasis on information from later transitions. The special case in which $\lambda = 0$, TD(0), uses the update

$$J^{k+1}(\mathrm{s_i}) = J^k(\mathrm{s_i}) + \alpha \left(c_{i,i+1}(\mu(\mathrm{s_i})) + \gamma J^k(\mathrm{s_{i+1}}) - J^k(\mathrm{s_i}) \right), \tag{4.36}$$

which is equivalent to a stochastic approximation of Bellman's equation (4.29) applied to a fixed policy.

Once a policy has been evaluated in a model-free manner using TD(λ), a policy improvement step needs to be applied. Unfortunately, to improve the policy in the manner suggested by Equation (4.33) a system model with known probabilities and costs is required. If a model is available, of course, one may be able to use value iteration instead of TD(λ); problems exist, though, in which a simulation-based, incremental policy evaluation method may still be desirable. In particular, trajectory-based incremental policy evaluation opens up the opportunity for a variation on the policy iteration algorithm described above; the policy update step can be performed before the policy evaluation algorithm (TD(λ)) has converged. The extreme case, in which μ is updated every time the estimate of J^{μ} is updated, is sometimes called optimistic policy iteration. The optimistic TD(0) algorithm is guaranteed to converge, but only specific variants of optimistic TD(1) are guaranteed to converge. Standard policy iteration, in which the policy is only updated after the TD(λ) evaluation has converged, converges for all λ.

An alternative model-free, simulation-based option for policy iteration is provided by Q-learning. In analogy with the optimal cost-to-go function $J^*(\mathrm{s_i})$, we can define the optimal Q-factor, $Q^*(\mathrm{s_i}, \mathrm{v})$, as the minimum expected cost-to-go from state $\mathrm{s_i}$ if action $\mathrm{v} \in \mathcal{V}(\mathrm{s_i})$ is chosen. Then we have:

$$\begin{aligned} Q^*(\mathrm{s_i}, \mathrm{v}) &= \mathrm{E}[c_{\mathrm{s_i},\mathrm{s_{i+1}}}(\mathrm{v}) + \gamma J^*(\mathrm{s_{i+1}})] \\ J^*(\mathrm{s_i}) &= \min_{\mathrm{v}\in\mathcal{V}(\mathrm{s_i})} Q^*(\mathrm{s_i}, \mathrm{v}) \end{aligned} \tag{4.37}$$

If the system model were known, basic value iteration could be performed on the Q-factors:

$$Q^{k+1}(\mathrm{s_i}, \mathrm{v}) = \sum_{\mathrm{s_j}} p_{i,j}(\mathrm{v})(c_{i,j}(\mathrm{v}) + \gamma \min_{\mathrm{v'}\in\mathcal{V}(\mathrm{s_j})} Q^k(\mathrm{s_j}, \mathrm{v'})) \tag{4.38}$$

The model-free Q-learning method is a simulation-based method analogous to TD(0):

$$\begin{aligned} Q^{k+1}(\mathrm{s_i}, \mathrm{v}) &= Q^k(\mathrm{s_i}, \mathrm{v}) + \alpha d_k \\ d_k &= c_{i,i+1}(\mathrm{v}) + \gamma \left(\min_{\mathrm{v'}\in\mathcal{V}(\mathrm{s_{i+1}})} Q^k(\mathrm{s_{i+1}}, \mathrm{v'}) \right) - Q^k(\mathrm{s_i}, \mathrm{v}) \end{aligned} \tag{4.39}$$

If every $(\mathrm{s_i}, \mathrm{v})$ pair is visited infinitely often and α is decreased appropriately, the Q-learning algorithm will converge to Q^*. Note that to guarantee that every $(\mathrm{s_i}, \mathrm{v})$ pair is visited infinitely often, some exploration mechanism often has to be built into the action selection, especially if the system is deterministic or nearly so; typically, instead of always picking the v that minimizes $Q^k(\mathrm{s_i}, \mathrm{v})$ (the greedy policy), a small proportion of the time a random v

is selected. If the randomization is chosen such that the product of the state and action spaces is fully explored, and the percentage of random control choices is decreased over time, the Q-factors converge to Q^* and the greedy policy converges to the optimal policy.

All of the preceding algorithms apply to J or Q specified by a lookup table. A lookup table representation is not always possible, however, in particular for large or continuous state spaces. In this case, the J or Q functions can be approximated by various means. For example, a function approximator depending on a vector of parameters or weights $\mathbf{w}$ can iteratively learn the optimal value or Q function. Variants of reinforcement learning which use function approximators are sometimes called neuro-dynamic programming [11], [10]. In the function approximator context, the TD(1) algorithm can be seen as a gradient method for minimizing the squared error between J^μ and an approximator $\tilde{J}(\mathrm{s_i}, \mathbf{w})$ over a simulated trajectory. We have:

$$\begin{aligned} E &= \frac{1}{2}\sum_i (\tilde{J}(\mathrm{s_i}, \mathbf{w}) - J^\mu(\mathrm{s_i}))^2 && (4.40) \\ \frac{\partial E}{\partial \mathbf{w}} &= \sum_i \frac{\partial \tilde{J}}{\partial \mathbf{w}}(\mathrm{s_i}, \mathbf{w}) \left(\tilde{J}(\mathrm{s_i}, \mathbf{w}) - J^\mu(\mathrm{s_i})\right) \\ &= \sum_i \frac{\partial \tilde{J}}{\partial \mathbf{w}}(\mathrm{s_i}, \mathbf{w}) \left(\tilde{J}(\mathrm{s_i}, \mathbf{w}) - \sum_{j=i}^{\infty} \gamma^{j-i} c_{j,j+1}(\mu(\mathrm{s_j}))\right) && (4.41) \end{aligned}$$

The TD(λ) weight update for general λ (the offline version, performed after the entire trajectory has been simulated) is given by:

$$\begin{aligned} \mathbf{w}^{k+1} &= \mathbf{w}^k - \alpha \sum_{i=0}^{\infty} \frac{\partial \tilde{J}}{\partial \mathbf{w}}(\mathrm{s_i}, \mathbf{w}^k) \left(\tilde{J}(\mathrm{s_i}, \mathbf{w}^k) - \sum_{j=i}^{\infty} (\gamma\lambda)^{j-i} c_{j,j+1}(\mu(\mathrm{s_j}))\right) \\ &= \mathbf{w}^k + \alpha \sum_{i=0}^{\infty} \frac{\partial \tilde{J}}{\partial \mathbf{w}}(\mathrm{s_i}, \mathbf{w}^k) \sum_{j=i}^{\infty} (\gamma\lambda)^{j-i} d_j && (4.42) \\ d_j &= c_{j,j+1}(\mu(\mathrm{s_j})) + \gamma \tilde{J}(\mathrm{s_{j+1}}, \mathbf{w}^k) - \tilde{J}(\mathrm{s_j}, \mathbf{w}^k) \end{aligned}$$

Only for $\lambda = 1$ does this update actually correspond to an iterative minimization of the squared error in Equation (4.40), however. For TD(0), the update looks like:

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \alpha \sum_{i=0}^{\infty} \frac{\partial \tilde{J}}{\partial \mathbf{w}}(\mathrm{s_i}, \mathbf{w}^k) d_i \qquad (4.43)$$

or, for the online version:

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \alpha \frac{\partial \tilde{J}}{\partial \mathbf{w}}(\mathrm{s_k}, \mathbf{w}^k) d_k \qquad (4.44)$$

If the approximation architecture for $\tilde{J}$ is linear in the weights $\mathbf{w}$, the TD(λ) iteration (both the offline and online versions) will converge, under some assumptions on the simulation method in addition to the usual requirements for α and state visitation. The limit of $\tilde{J}$ may depend on the value of λ, though. With a nonlinear approximation architecture, TD(λ) cannot be proved to converge, and in fact may diverge. There have been some successes with neural network approximators, though; the most notable is Tesauro's TD-Gammon backgammon player [97], [98].

Optimistic policy iteration can be performed with function approximators, as well, but its convergence properties are not well understood. In particular, convergence of the weights $\mathbf{w}$ is not always accompanied by convergence of the policy μ. Again, these methods require knowledge of the transition probabilities and costs.

Finally, Q-learning can also be performed with a function approximator $\tilde{Q}(\mathbf{s_i}, \mathbf{v}, \mathbf{w})$. The updates are similar to the TD(0) updates in Equations (4.43)–(4.44).

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \alpha \sum_{i=0}^{\infty} \frac{\partial \tilde{Q}}{\partial \mathbf{w}}(\mathbf{s_i}, \mathbf{v_i}, \mathbf{w}^k) d_i \tag{4.45}$$

or, for online updating:

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \alpha \frac{\partial \tilde{Q}}{\partial \mathbf{w}}(\mathbf{s_k}, \mathbf{v_k}, \mathbf{w}^k) d_k \tag{4.46}$$

In general, this algorithm cannot be guaranteed to converge, though there are results for very restricted special cases.

More detailed discussions about dynamic programming, reinforcement learning, temporal difference algorithms, and neuro-dynamic programming can be found in [11], [96], [84], [53], and [7].

4.2.3 Learning Implementation

In the diver problem, although the dynamics are deterministic, we do not have a model of the system available, so we are forced to use the Q-learning form of reinforcement learning. The state vector $\mathbf{s} = [\mathbf{s_d}, \mathbf{s_s}]$ is partially continuous, and the action vector $\mathbf{v} = [t_s]$ is continuous as well, so a function approximator is required. In the controller design shown in Figure 4.4, the coordinator network, a multilayer perceptron (a nonlinear architecture), serves as an approximator or internal model of $Q(\mathbf{s}, \mathbf{v})$, the smallest dive cost that can be accrued from state $\mathbf{s}$ with action $\mathbf{v}$. A neural network is a reasonable choice for the coordinating controller, given the large number of inputs required; for feature-based methods, like radial basis functions, the number of parameters is exponential in the number of inputs. Neural networks have a more compact representation and also give a smoother approximation, as mentioned earlier (Section 4.1.2).

Unfortunately, training one network on all three dives (jackknife, one-and-one-half-somersault pike, and one-and-one-half-somersault, full-twist) proved very difficult. With the input representation used here, the twisting dive and the two non-twisting dives tended to interfere with each other. Various other representations were tested, but none seemed to improve the situation. Thus, the controller was trained with two separate coordinator networks, one for the jackknife and one-and-one-half-somersault pike dives and one for the twisting dive. Each of these coordinator networks takes as input the state $\mathbf{s} = [\mathbf{s}_d, \mathbf{s}_s]$ and action $\mathbf{v} = [t_s]$ vectors and outputs the Q approximation $\tilde{Q}(\mathbf{s}, \mathbf{v}, \mathbf{w})$. With separate networks, some of the inputs in each network no longer carry any information (ϕ_{td} is a constant for each net, for example) but were retained so the two networks would have the same inputs and so that possible future expansion of the dive set would be more straightforward. Condensing the number of inputs for the two networks did not seem to improve performance, in any case. It is possible that using a separate network for each behavior within each dive might improve performance further; the Q function representations for the pike and layout behaviors in the twisting dive coordinator network also sometimes interfered with each other, although to a lesser extent.

Each coordinator network input is scaled to lie in [0, 1] using an affine transformation on the range shown in Table 4.5. Since there are only two body positions, corresponding to completed multi-joint synergies, which can figure as inputs to the network, $\mathbf{p}$ has only one element. $\mathbf{p} = [0]$ corresponds to the pike position, and $\mathbf{p} = [1]$ corresponds to the throw position. Each of the two coordinator networks has two hidden layers containing 24 units each. Significantly smaller networks did not train well. The output neuron in each net uses a linear activation function, and the others use the sigmoid function in Equation (4.6).

At the end of a dive, the performance is assigned a cost:

$$C = .01 \left[k_s(\phi_s - \phi_{sd})^2 + k_t(\phi_t - \phi_{td})^2 + k_c\phi_c^2 + k_r\dot{\phi}_t^2 + \sum_{\text{DOF}} (\theta - \theta_d)^2 + \dot{\theta}^2 \right], \qquad (4.47)$$

where ϕ_s is the somersault angle of the body, ϕ_t is the twist angle of the body, and ϕ_c is the cartwheel (roll) angle of the body (which ideally should always be zero). The scaling

Parameter	Range
ϕ_{sd}	$[0, 15.7]$
ϕ_{td}	$[0, 6.2832]$
ϕ_s	$[0, 15.7]$
ϕ_t	$[0, 6.2832]$
t	$[0, 2]$
$\mathbf{p}$	$[0, 1]$
$\mathbf{v} = [t_s]$	$[0, 2]$

Table 4.5: Scaling ranges for coordinator network inputs. Each parameter value was scaled with an affine transformation so that the range shown in the table corresponds to $[0, 1]$. Angles are in radians and times are in seconds.

constants were given the values $k_s = 100$, $k_t = 25$, $k_c = 25$, and $k_r = 25$. No costs are assigned at any other step of the dive state machine. In general, this dive evaluation could have other terms from penalties incurred by bad requests to single-DOF controllers or failure to finish the dive, for example. Other forms of dependence on somersault and twist error, for example, could be used, as well. Also, additional costs could be assigned at intermediate stages of the dive. The cost function shown in Equation (4.47) is simply one possibility that works as a reinforcement signal.

The action selector shown in Figure 4.4 uses the current $\tilde{Q}$ function output by the coordinator network to determine what action vector $\mathbf{v}$ to send to the next multi-joint synergy. During learning, the selector chooses a random action some of the time, in order to better explore the control space. When not choosing the action randomly, the action selector picks the greedy action, the action which minimizes the coordinator network's Q output. This minimization is performed using the quasi-Newton method described in Section 4.1.2. For this application, of course, it is not the weights in the network that are being varied, but the vector $\mathbf{v}$, to minimize the network output Q. The gradient $\frac{\partial Q}{\partial \mathbf{v}}$ is calculated using backpropagation; considering the output neuron, the activation of an input neuron specifying a component of the action vector v_j, and the neurons in the first hidden layer (labeled with the subscript i), we have

$$\begin{aligned}\frac{\partial Q}{\partial v_j} &= \frac{\partial o}{\partial v_j}\\ &= \sum_i \frac{\partial o}{\partial n_i}\frac{\partial n_i}{\partial v_j}\\ &= \sum_i \delta_i w_{ij} \qquad (4.48)\end{aligned}$$

$$\delta_i = \frac{\partial o}{\partial n_i} \qquad (4.49)$$

The definition (4.49) here is only different from the standard one in Equation (4.17) by a factor of e, the error at the output unit. Thus, by simply defining the "error" at the output unit to be 1, the same backpropagation algorithm as was used in Section 4.1.2 can be used here to calculate $\mathbf{g} = \frac{\partial Q}{\partial \mathbf{v}}$. This gradient can then be used in the BFGS update formula, Equations (4.13)–(4.14), applied to the action vector $\mathbf{v}$ instead of the network weights $\mathbf{w}$. The coefficient α is calculated by the line search algorithm described in Section 4.1.2. As the Q function is likely to have multiple local minima as well as some fairly flat areas, the function minimization is performed with 25 initial conditions for $\mathbf{v} = [t_s]$ randomly generated in $\mathbb{R}$, and the best result is chosen. The flatness of the Q function also makes it necessary to modify the line search algorithm slightly, so that if the absolute value of either the numerator or the denominator in the expression for α_4, Equation (4.23), is less than 10^{-12}, the new point α_4 is chosen as $\frac{\alpha_1+\alpha_3}{2}$. Values smaller than this cutoff are too close to the precision limit of the machine for α_4 and the bound on the percentage error (see Section 4.1.2) to be computed accurately enough.

Learning by Watching

As discussed in Section 4.2.1, the "learning by watching" stage of learning consists of defining the dive's cost structure (in the MDP sense), defining the multi-joint synergies and the order in which they are performed, and roughly initializing the choice of parameters $\mathbf{v}$ sent to each multi-joint synergy. For the diver, this latter part of the "learning by watching" stage consisted of initializing the network weights and thus $\tilde{Q}$. This initial training for each coordinator network used a set of synergy timing data, shown in Table 4.6, obtained from video of the 1996 men's and women's Olympic platform diving competitions. Although this data was obtained by human estimation using a frame-by-frame examination of the video, in the future such "learning by watching" kinematic and timing data could be extracted from video automatically [13]. As can be seen from Table 4.6, the Olympic divers showed quite a bit of variation in the timing of their dives. Some of the variation is probably due to differences in their initial angular momenta at takeoff; some is due to differences in technique (the United States and Chinese divers used very different styles, for example). Both men's and women's timing data was included to make a larger sample set. There was only one jackknife example available, so the "learning by watching" training for that dive was much less meaningful than for the others. The data in Table 4.6 is also fairly noisy because the low frame rate of standard video (30 frames/sec) relative to the speed of the divers' movements made it difficult to estimate the beginning and end times of the multi-joint synergies.

The data were converted into two sets of positive $(\mathbf{s}, \mathbf{v})$ instances (one for the twisting dive and one for the other two dives) and assigned Q values of 0. Then, for each dive specification $[\phi_{sd}, \phi_{td}]$ and position vector $\mathbf{p}$, 100 sets of random values of ϕ_s, ϕ_t, t, and $\mathbf{v} = [t_s]$ were generated. These negative instances were all assigned large positive Q values equal to the square of the distance to the nearest positive point (restricted to the same $[\phi_{sd}, \phi_{td}]$ and $\mathbf{p}$) multiplied by 5 for the twisting dive or by .1 for the jackknife and somersault dives. The coordinator networks were then trained using the quasi-Newton method (see Section 4.1.2) on the entire data set of positive and negative instances, to a cutoff total squared error of .2 for the twisting dive or .1 for the jackknife and somersault dives. Figures 4.5–4.7 show simulation frames of the resulting dives. The Olympic video data proved surprisingly good for training the diver; with lower training cutoffs than these, the "learning by watching" training was so effective that "learning by doing" was hardly needed. The more loosely trained networks described here allowed the capabilities of the "learning by doing" stage to be demonstrated.

The "learning by watching" training stage is essential for the diver problem. The extreme nonlinearity of the Q function creates multiple local minima, even with only one action parameter t_s, in which the Q-learning algorithm can get stuck. The "learning by watching" $\tilde{Q}$ initialization ameliorates this problem by restricting the search space to a region that should be near the global minimum. Furthermore, the "learning by watching" weight initialization gives the Q-learning algorithm something of a head start, so that it requires many fewer iterations to converge than it would starting with randomized network weights. Since the Q-learning algorithm requires time-intensive simulation for each iteration,

forward pike (jackknife)			
	pike		layout
Diver	t_{end}	ϕ_s	t_{start}
Xiao	.400	1.745	1.100

$1\frac{1}{2}$ somersault pike			
	pike		layout
Diver	t_{end}	ϕ_s	t_{start}
Clark	.333	2.967	1.067
Guo	.433	3.142	1.100
Fu	.400	2.793	1.167
Vyguzova	.400	3.316	.967
Walter	.333	3.142	.967
Platas	.400	3.142	1.267
Tian	.333	2.356	1.367
Akhmetbekov	.400	3.142	1.233
Pichler	.333	3.142	1.200
Timoshinin	.467	3.490	1.200
Sautin	.367	2.793	1.267
Jeffrey	.333	2.618	1.133

$1\frac{1}{2}$ somersault 1 twist							
	throw			pike			layout
Diver	t_{end}	ϕ_s	ϕ_t	t_{start}	t_{end}	ϕ_s	t_{start}
Guo	.367	1.571	1.571	.600	.900	6.109	1.067
Fu	.233	1.047	1.571	.567	.867	5.760	1.033
Clark	.467	1.920	1.920	.767	1.033	6.283	1.200
Jeffrey	.467	1.745	3.142	.667	.933	5.236	1.267
Tian	.467	1.920	1.920	.733	1.067	5.585	1.267
Xiao	.333	1.222	1.396	.700	.967	5.061	1.433
Akhmetbekov	.433	2.094	1.396	.700	1.000	5.934	1.200
Pichler	.533	3.142	1.571	.767	1.000	5.934	1.200

Table 4.6: Coordinator network initialization data obtained from Olympic men's and women's platform diving video. t_{start} and t_{end} refer to start and end times of the specified synergy, in seconds. ϕ_s and ϕ_t specify the somersault and twist angle at the end of the synergy, in radians. Some divers performed their twists in a positive direction and some in a negative direction; the values in the table have been normalized for positive twist. ϕ_t values not specified in the table are zero for the jackknife and one-and-one-half-somersault pike; ϕ_t at the end of the pike in the twisting dive is 2π. The start time t_{start} for the first behavior in a dive always was zero, and the initial angles were always used as $\phi_s = .175$ and $\phi_t = 0$, as discussed in Section 4.2.1, though the start times and angles varied some among the divers. The divers listed above the horizontal bar in the second and third tables are women, and those listed below are men.

this advantage can be quite significant.

Learning by Doing

After the "learning by watching" initialization, the coordinator network refined its model of the Q function through "learning by doing." In this learning phase, the controller tested choices of $\mathbf{v}$ through simulation and used the resulting costs to update the network weights with the Q-learning algorithm.

In the "learning by watching" stage, the synergy activation sequence was fixed for each dive. As discussed in Section 4.2.1, the controller behaves like a timed state machine as it switches between the multi-joint behavioral synergies. The jackknife and the one-and-one-half-somersault pike both use the sequence (pike, layout). The twisting dive uses the sequence (throw, pike, layout). This state machine becomes the Markov decision process underlying the Q-learning algorithm.

At each transition of the behavioral state machine, the controller's action selector uses the coordinator network's model of Q to choose the parameter vector for the next synergy. The coordinator network's approximation is updated at each transition as well. A flow chart showing the linked processes of action selection, simulation, and learning is shown in Figure 4.8. At the beginning of a dive, at $t = 0$, the first synergy in the state machine is initiated. This synergy uses the initial values of θ and $\dot{\theta}$ for each DOF to determine the inputs $\mathbf{y}_{\text{d}}$ for the single-DOF controllers. After the synergy has executed, the resulting state $\mathbf{s}_{\text{s}} = [\phi_s, \phi_t, t, \mathbf{p}]$ is input to the coordinator network, and the action selector chooses an action vector (possibly random) for the next synergy (see Figure 4.8). After waiting for the amount of time specified by $\mathbf{v} = [t_s]$, the new synergy uses the current θ and $\dot{\theta}$ for each

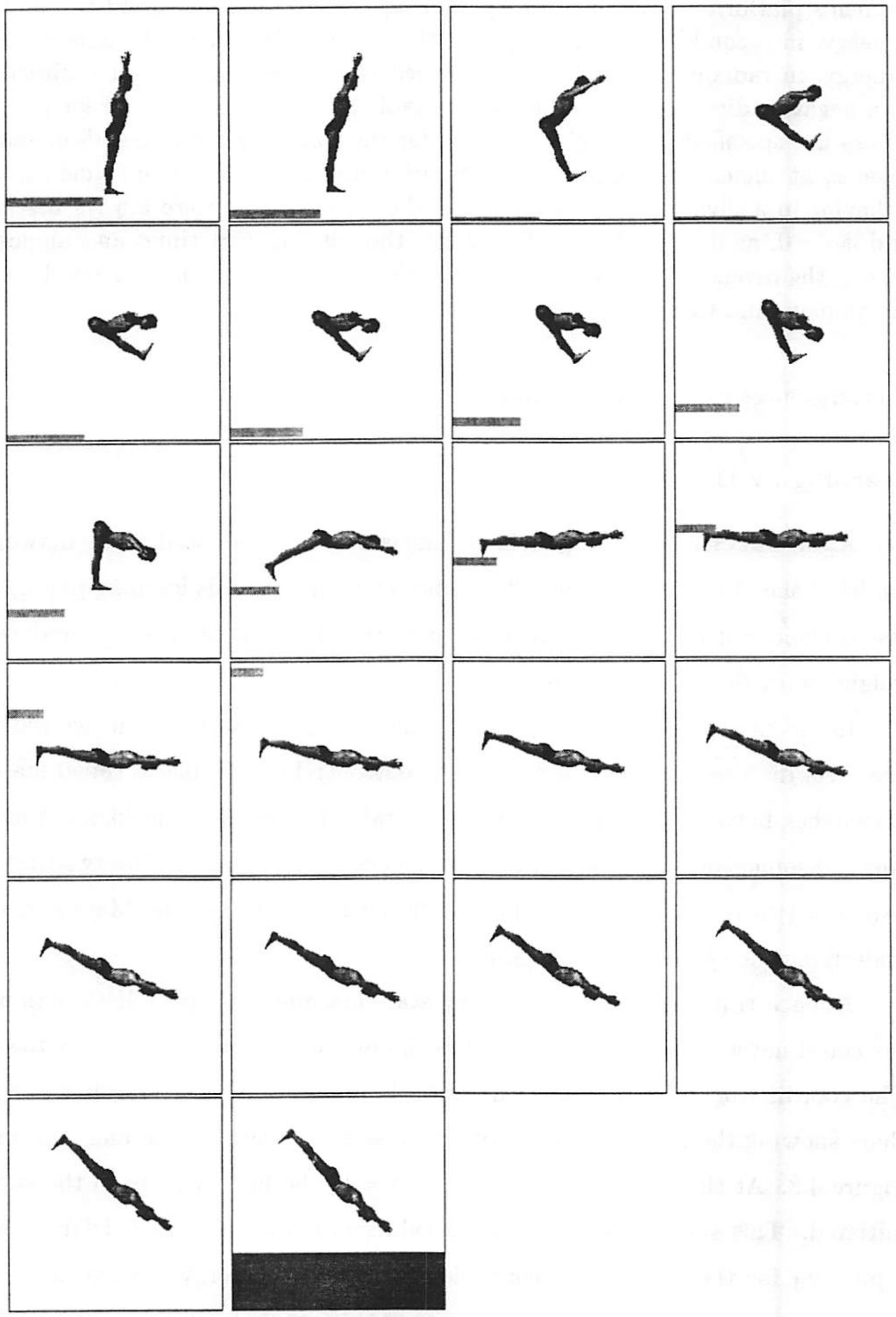


Figure 4.5: Frames from a simulation of the jackknife dive after the "learning by watching" stage but before the "learning by doing" stage. The diving platform is virtual and does not affect the trajectory.

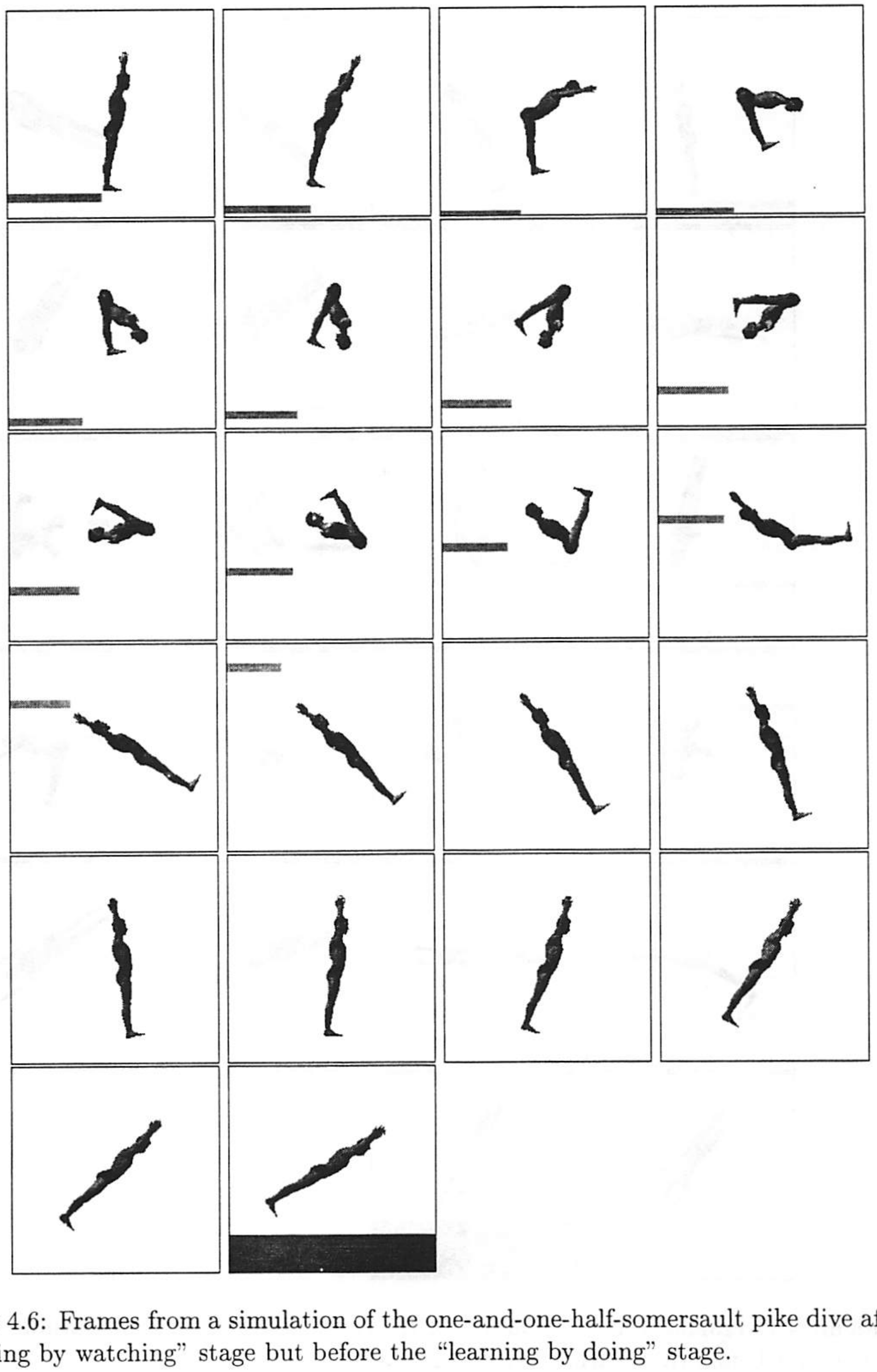


Figure 4.6: Frames from a simulation of the one-and-one-half-somersault pike dive after the "learning by watching" stage but before the "learning by doing" stage.

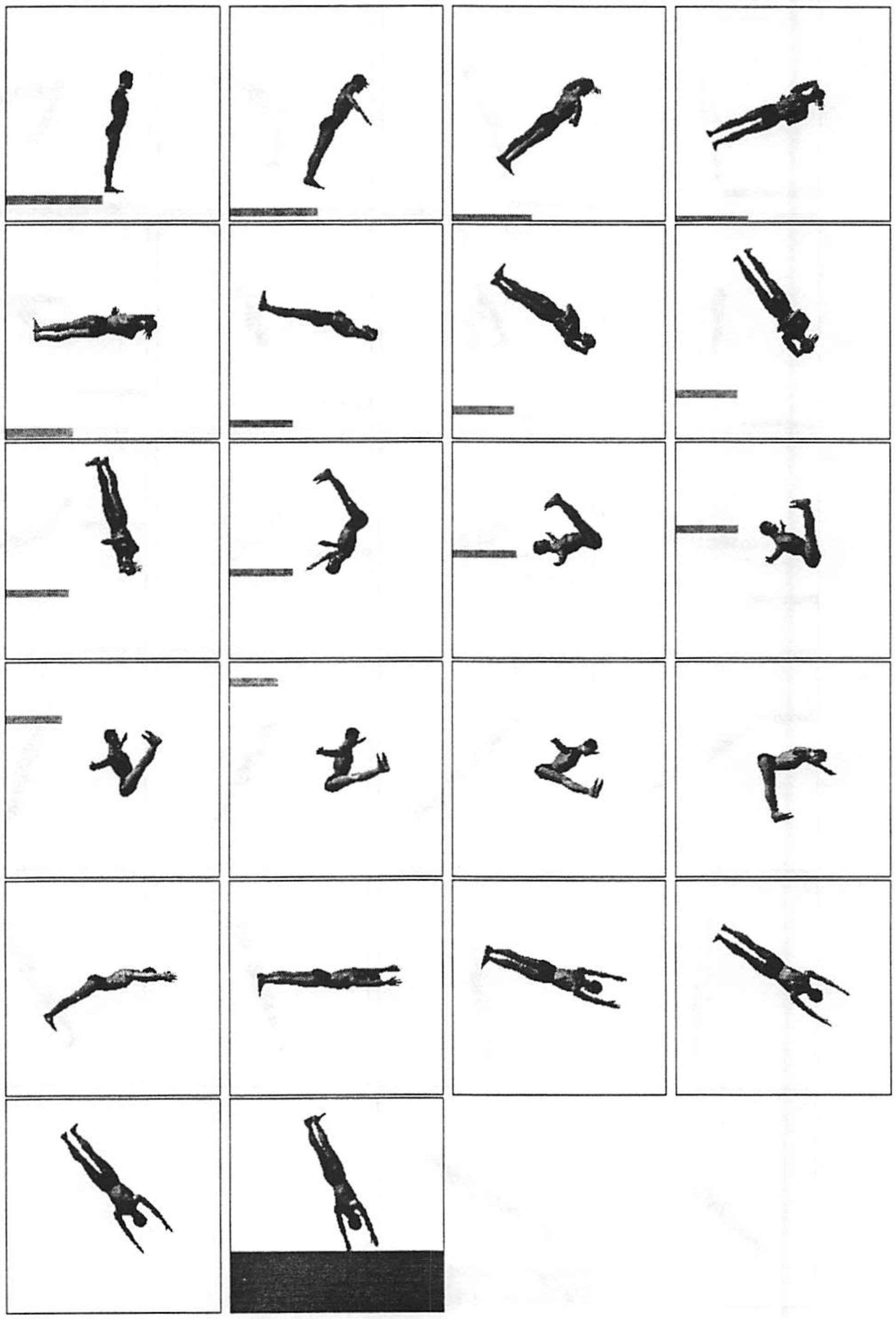


Figure 4.7: Frames from a simulation of the twisting dive after the "learning by watching" stage but before the "learning by doing" stage.

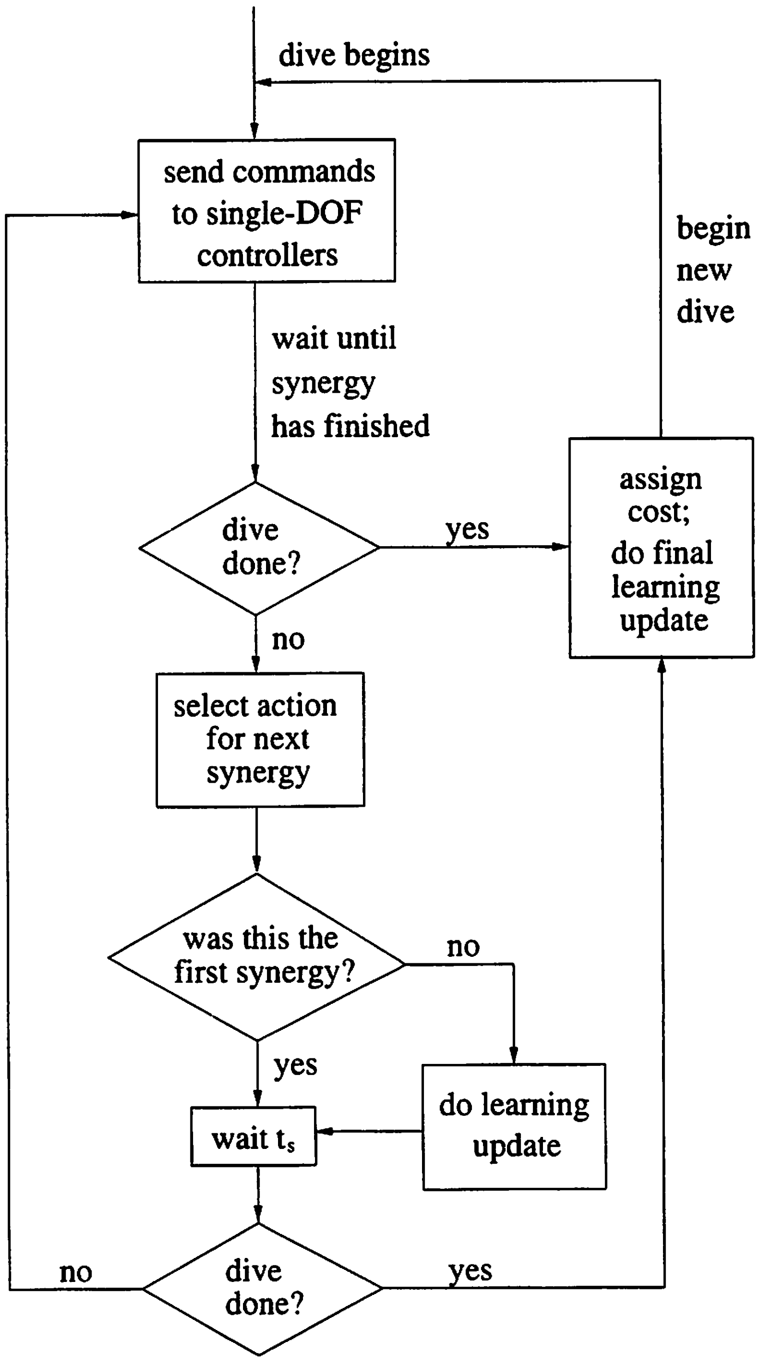

dive begins
send commands to single-DOF controllers
begin new dive
wait until synergy has finished
assign cost; do final learning update
dive done?
yes
no
select action for next synergy
was this the first synergy?
no
yes
do learning update
wait t_s
no
dive done?
yes

Figure 4.8: Flow chart showing the relationships between the processes of action selection, simulation, and learning during the "learning by doing" stage.

DOF to determine the new inputs $\mathbf{y_d}$ for the single-DOF controllers. At any stage, if there are no more synergies left for the dive being executed, the simulation simply continues until the diver reaches the water. While waiting for a synergy's start time or after the entire sequence of behaviors has been executed, the joints are held in position by the PD controllers mentioned in Section 4.2.1. Each PD controller's set point is defined by the DOF's position at the end of its previous movement.

The Q-learning algorithm is applied to the coordinator net at each transition between synergies. The trajectories in the diver system are finite, ending when the diver hits the water, so the sum in Equation (4.45) is finite. With these short trajectories, no discounting is needed, so $\gamma = 1$. When one synergy finishes, the best action vector $\mathbf{v}$ is computed for the next synergy, as described above. The Q value corresponding to this action vector is then used to compute the temporal difference:

$$d_k = \min_{\mathbf{v}' \in \mathcal{V}(\mathbf{s_{k+1}})} \tilde{Q}(\mathbf{s_{k+1}}, \mathbf{v}', \mathbf{w}^k) - \tilde{Q}(\mathbf{s_k}, \mathbf{v}, \mathbf{w}^k), \tag{4.50}$$

where $\mathbf{s_k}$ is the state at the beginning of the previous synergy and $\mathbf{s_{k+1}}$ is the state at the start of the new synergy. If the synergy is the last in the dive, no new behavior is computed, and the temporal difference is computed as:

$$d_k = C - \tilde{Q}(\mathbf{s_k}, \mathbf{v}, \mathbf{w}^k) \tag{4.51}$$

where C is the cost assigned at the end of the dive (Equation (4.47)). If the temporal difference d_k is considered to be the negative of the pattern error e on the coordinator network output neuron, with $E_p = \frac{1}{2}e^2$, this error can be propagated back through the network as described in Section 4.1.2 (Equations (4.16)-(4.21)) in order to calculate the term $\frac{\partial E}{\partial \mathbf{w}} = \frac{\partial \tilde{Q}}{\partial \mathbf{w}}(\mathbf{s_k}, \mathbf{v}, \mathbf{w}^k) d_k$. In addition to the gradient term, it was useful as an exploration aid to add a small random component η to the weights at each step. The weight update is then:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \alpha \frac{\partial \tilde{Q}}{\partial \mathbf{w}}(\mathbf{s_k}, \mathbf{v}, \mathbf{w}^k) d_k + \eta \tag{4.52}$$

The next action $\mathbf{v}$ is chosen by minimizing the Q function over all possible actions, as described in Section 4.2.1, except for some percentage of actions which are chosen randomly.

The coordinator network for the jackknife and one-and-one-half-somersault pike dives was trained over a total of 2905 dives (at each iteration, one of the two dives was randomly selected for simulation and learning). The network for the twisting dive was trained over

# iterations	α	$\eta_{\max}$	random v
90	.005	2.5×10^{-5}	.25
965	.005	2.5×10^{-4}	.25
650	.005	2.5×10^{-5}	.25
700	.05	2.5×10^{-5}	.25
300	.1	5.0×10^{-5}	.25
200	.05	2.5×10^{-6}	.10

Table 4.7: Training schedule for the one-and-one-half-somersault pike and jackknife dives. The first column gives the number of iterations, the second the learning rate α, the third the maximum absolute value of the noise η, and the fourth the fraction of the time random actions **v** were selected.

# iterations	α	$\eta_{\max}$	random v
210	.0001	2.5×10^{-4}	.25
165	.00001	2.5×10^{-5}	.25
700	.00001	2.5×10^{-6}	.25
600	.00001	2.5×10^{-7}	.10

Table 4.8: Training schedule for the twisting dive. The first column gives the number of iterations, the second the learning rate α, the third the maximum absolute value of the noise η, and the fourth the fraction of the time random actions **v** were selected.

1675 dives. The schedule of learning rates α, ranges of the noise term η, and percentages of random action choices are given in Tables 4.7 and 4.8, and the training results for the three dives are shown in Figures 4.9–4.25. For a random action choice, the $\mathbf{v} = [t_s]$ value was chosen uniformly in $[t_s^\dagger - .0005, t_s^\dagger + .0005)$, where $t_s^\dagger$ is the greedy action, the one that minimizes $\tilde{Q}$. At each weight update, η was chosen uniformly in $[-\eta_{\max}, \eta_{\max})$.

The trained controller produces good dives; the "learning by doing" phase improved significantly on the "learning by watching" phase. The twisting dive network training seemed to rely more heavily on having a good "learning by watching" initializaton than did the training of the network for the jackknife and one-and-one-half-somersault pike dives. The twisting dive is more difficult to learn both because it involves a sequence of two action choices rather than just one and because it involves transfers of rotational velocity between the somersault and twisting axes; the behavior of the system is very nonlinear and is sensitive to the time at which the diver comes out of the twist into the pike position.

At the end of the training, the action choices for all three dive types were still significantly different from the optimal values, however. The controller learns the Q value very accurately at the current greedy action, but, even with large amounts of randomization injected (both in the choice of $\mathbf{v}$ and in the weight update), does not seem to match this level of accuracy for values of t_s away from $t_s^\dagger$, and so is unable to finely locate the global minimum. The top panel in Figure 4.20, for example, reveals that at the end of training, after the temporal differences have converged to quite small values, randomized action choices still sometimes yield smaller errors than the action that minimizes the trained $\tilde{Q}$. This residual error is a reflection of the small final twist angle and velocity errors visible in Figures 4.21 and 4.22. Similar final errors can be observed in the other dives, as well. This phenomenon will be discussed further in Section 5.2.

The "learning by watching" initialization ensures that the Q-learning algorithm need not learn values very far away from the global minimum, but higher accuracy in a region around the greedy action is still necessary to achieve the smallest dive costs. Lower final dive errors can sometimes be achieved by training the network to a smaller cutoff during the "learning by watching" stage, as mentioned above; the dive error then starts out much lower in the "learning by doing" stage and therefore tends to converge to a lower final error more easily. In the twisting dive, the difficulty in achieving small dive costs is also partly due to the nonlinearity of the diver dynamics mentioned above: small errors in the time for pulling out of the twist into the pike tend to result in unstable rotation about the twist and cartwheel axes, which is exacerbated later by going into the layout position (see Figure 4.7, for example). In all three dives, the minimum achievable dive cost may also be limited by the accuracy of the single-DOF controllers (including the effectiveness of the low-level PD servos). These limitations could possibly be overcome through the use of global feedback, but such feedback would be difficult to design for a system with behavioral goals like the diver, as will be discussed in Chapter 5.

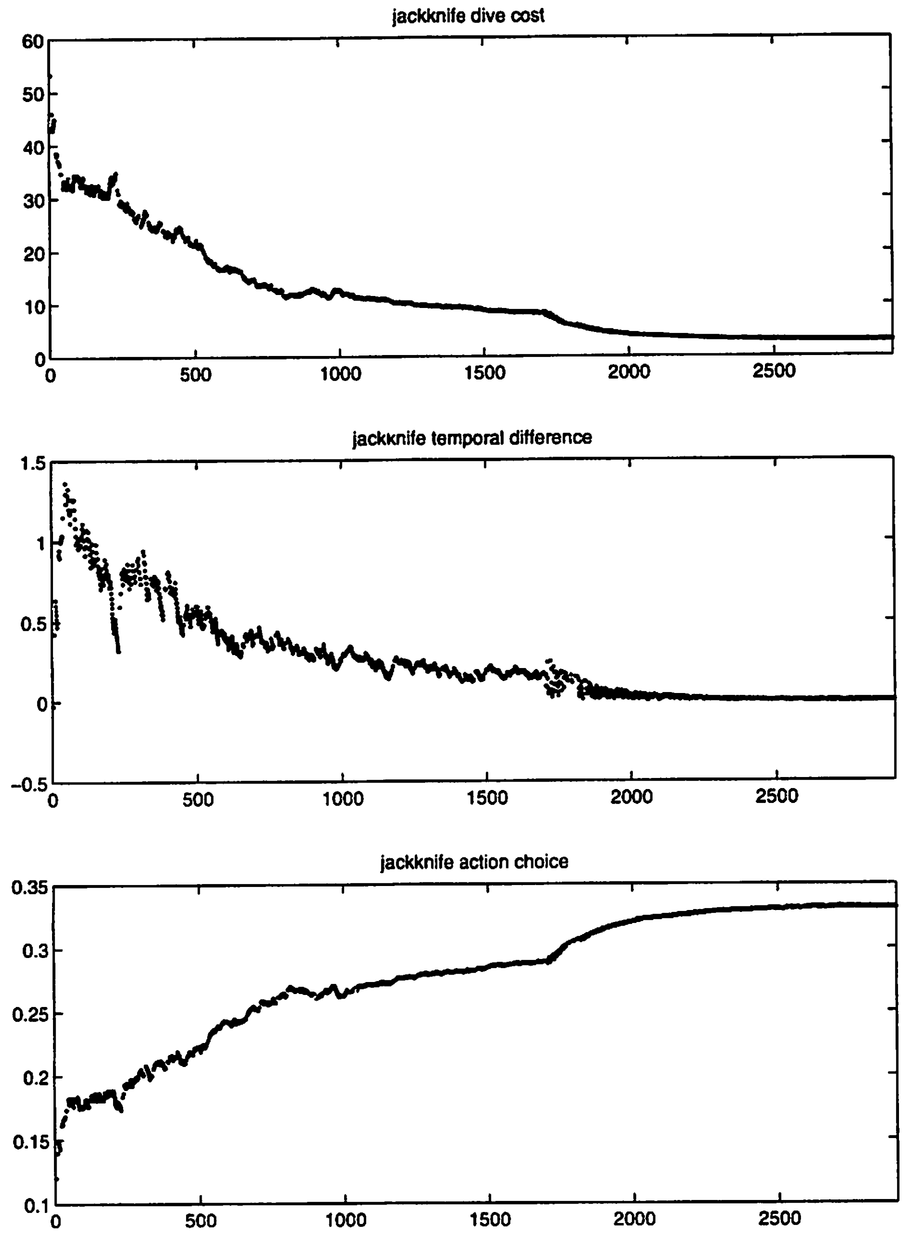


Figure 4.9: Training data for the jackknife dive. First plot: Dive costs, calculated as in Equation 4.47. Second plot: Temporal differences at the state transition where learning occurs (pike to layout). Third plot: Scaled action choices (t_s, the time to wait before executing the layout synergy). The unscaled t_s values (times, in seconds) are two times the plotted scaled values. The iteration number on the horizontal axis indicates the total number of jackknife and one-and-one-half-somersault pike trials; this figure and Figure 4.10 show interleaved learning trials.

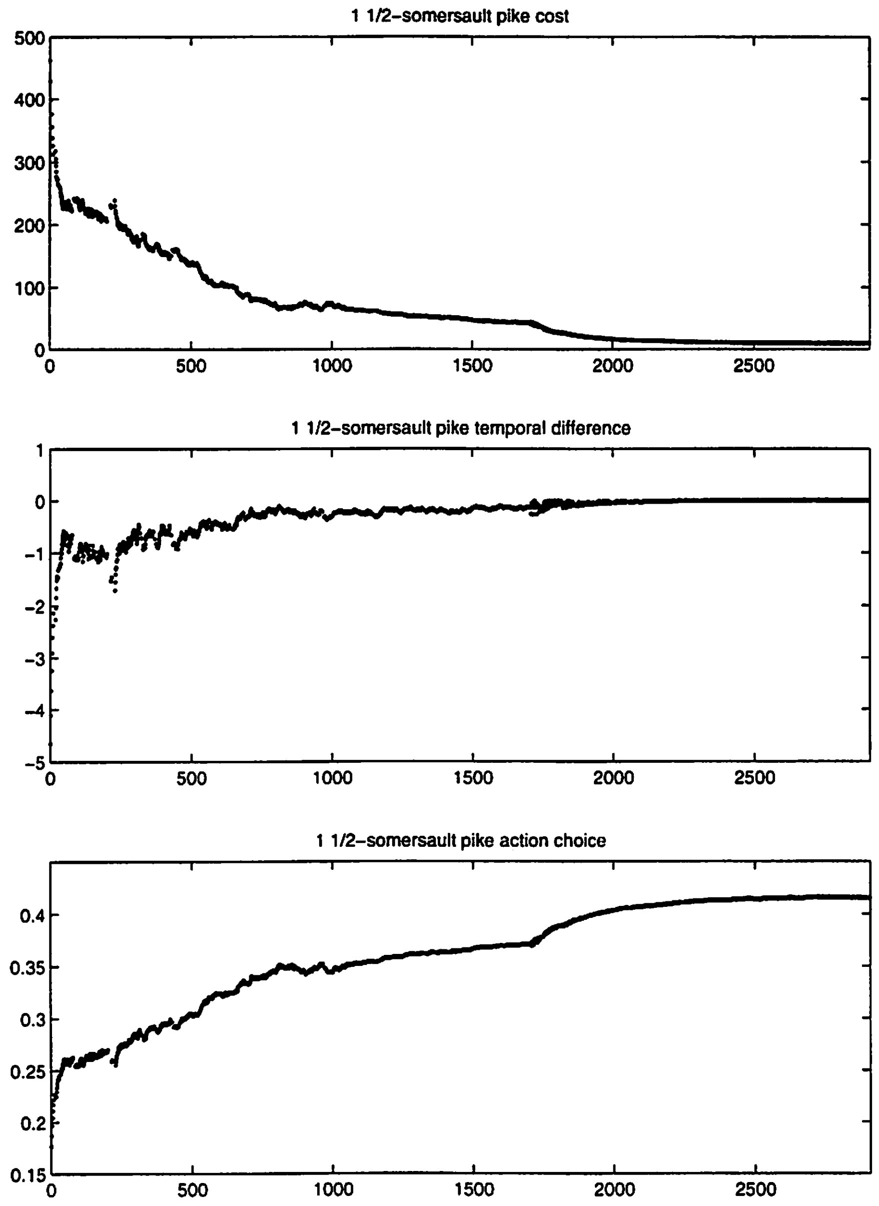


Figure 4.10: Training data for the one-and-one-half-somersault pike dive. First plot: Dive costs, calculated as in Equation 4.47. Second plot: Temporal differences at the state transition where learning occurs (pike to layout). Third plot: Scaled action choices (t_s, the time to wait before executing the layout synergy). The unscaled t_s values (times, in seconds) are two times the plotted scaled values. The iteration number on the horizontal axis indicates the total number of jackknife and one-and-one-half-somersault pike trials; this figure and Figure 4.9 show interleaved learning trials.

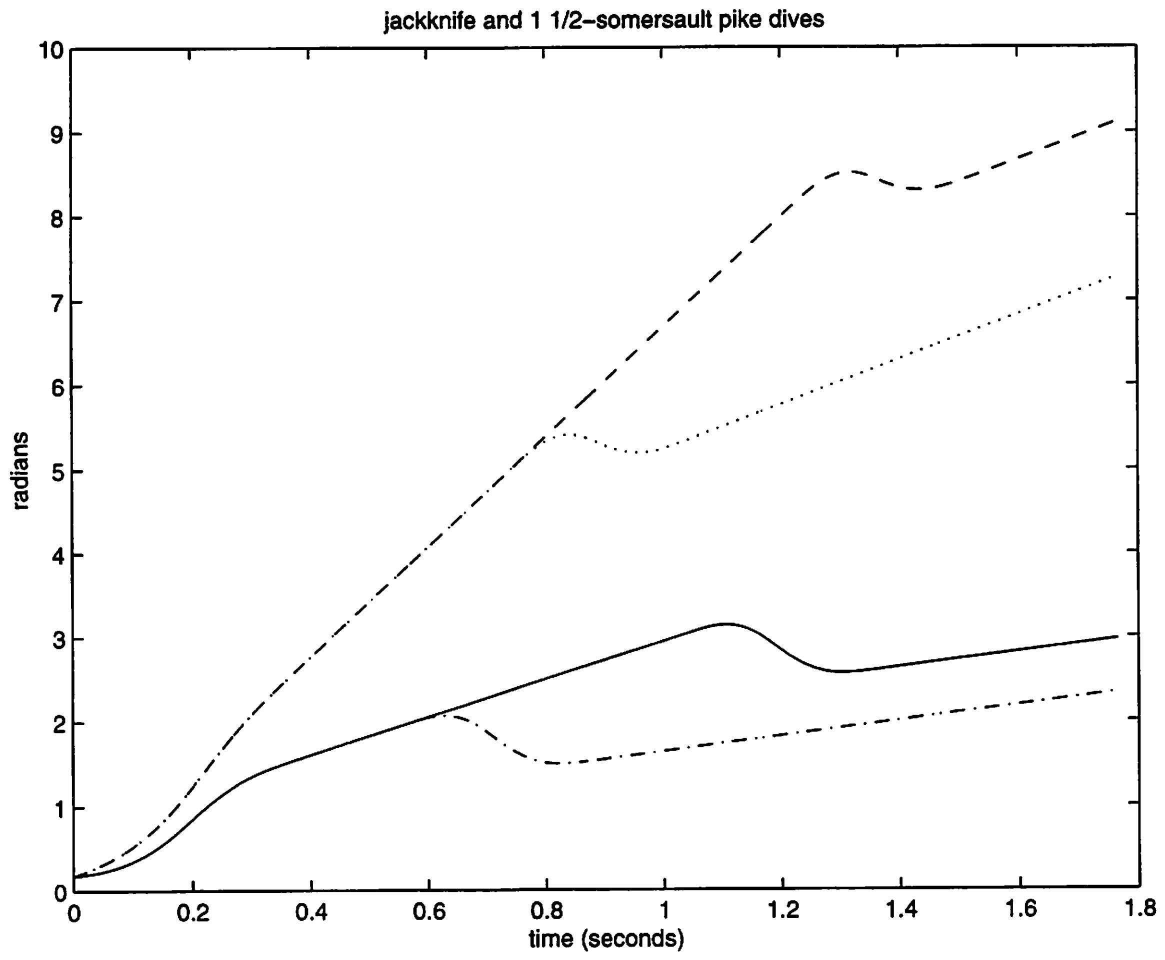


Figure 4.11: Comparison of the jackknife somersault dives after the "learning by watching" stage and after the full "learning by doing." The solid line indicates the somersault angle for the jackknife after the full learning, and the dashed line is the somersault angle for the somersault dive after the full learning; the dash-dot line is the jackknife after only the "learning by watching" phase, and the dotted line is the somersault after "learning by watching." All angles are in radians.

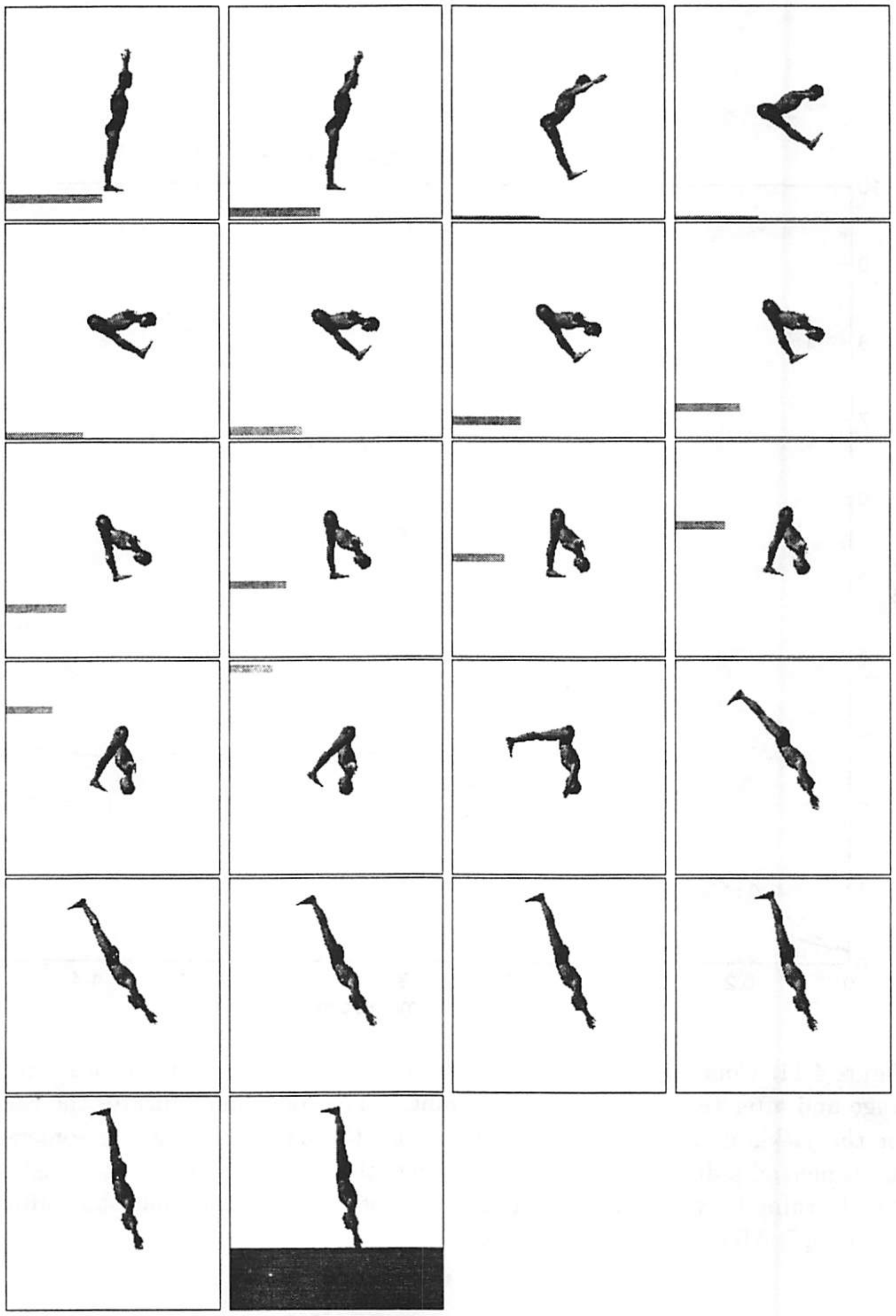


Figure 4.12: Frames from a simulation of the learned jackknife dive.

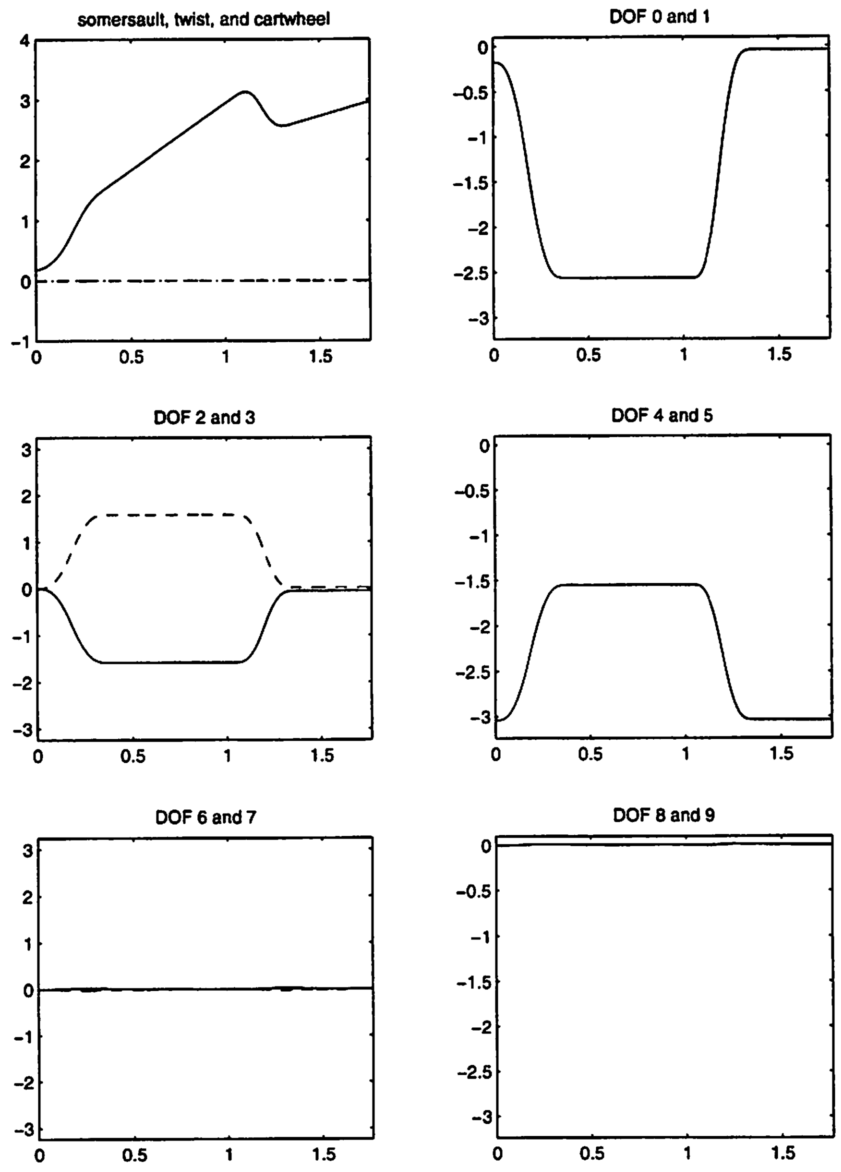


Figure 4.13: Body and joint angles (in radians) for the learned jackknife dive. The first plot shows the somersault angle. For the other five plots, the solid line is the even DOF, and the dashed line is the odd DOF. Degrees of freedom 0 and 1 are the right and left hip, 2, 4, and 6 are the three Euler angles of the right shoulder, 3, 5, and 7 are the angles of the left shoulder, and 8 and 9 are the right and left elbow. The horizontal axes are time, in seconds.

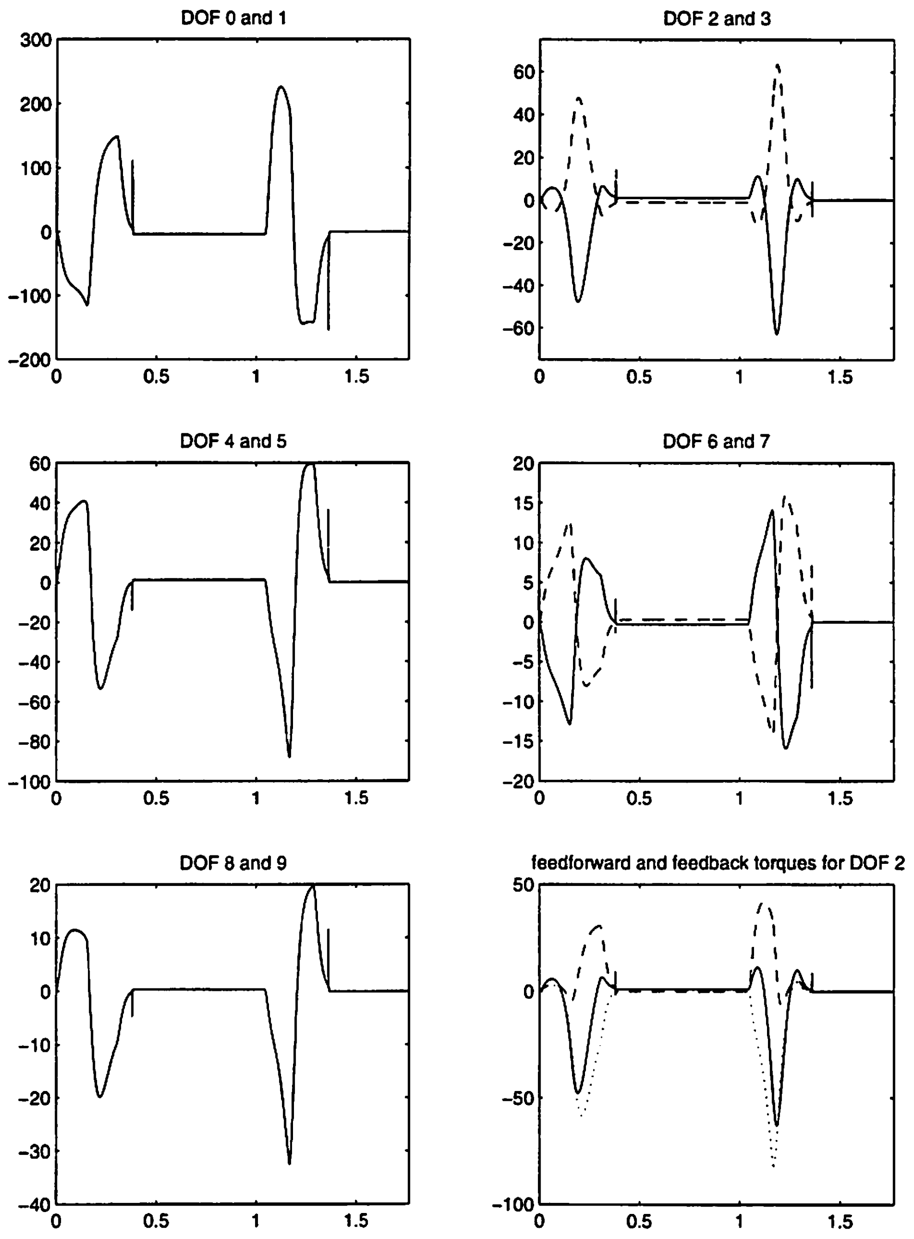


Figure 4.14: Torques (in Newton-meters) for the learned jackknife dive. In the first five plots, the solid line is the even DOF, and the dashed line is the odd DOF. The last plot shows the total torque (solid), the feedforward torque (dashed), and the feedback torque (both during and between movements; dotted) for DOF 2. Degrees of freedom 0 and 1 are the right and left hip, 2, 4, and 6 are the three Euler angles of the right shoulder, 3, 5, and 7 are the angles of the left shoulder, and 8 and 9 are the right and left elbow. The horizontal axes are time, in seconds.

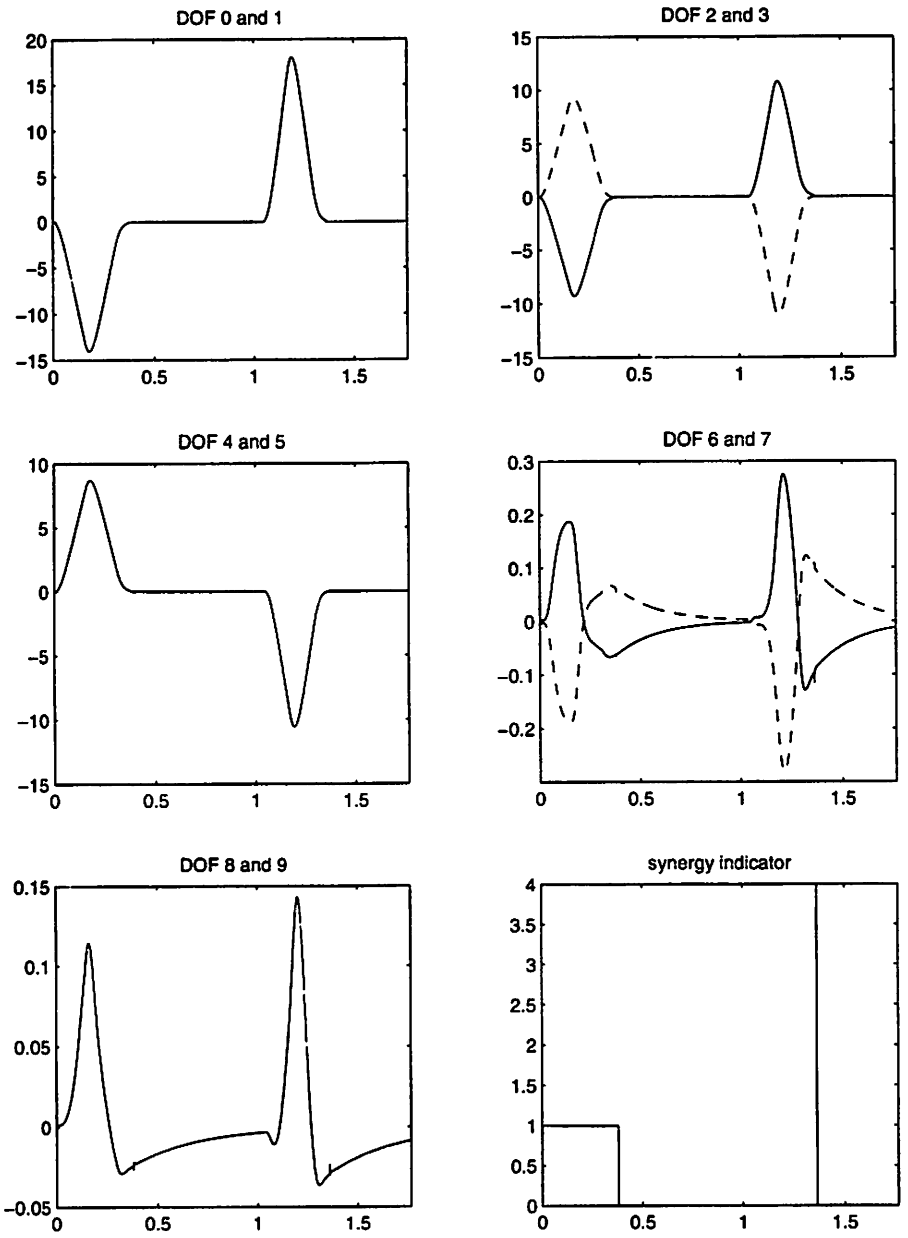


Figure 4.15: Velocities (in radians/second) for the learned jackknife dive. In the first five plots, the solid line is the even DOF, and the dashed line is the odd DOF. The last plot indicates which synergy is active at any given time: 0 represents the layout synergy, 1 represents the pike synergy, and 4 indicates the state machine has completed all the required synergies. Degrees of freedom 0 and 1 are the right and left hip, 2, 4, and 6 are the three Euler angles of the right shoulder, 3, 5, and 7 are the angles of the left shoulder, and 8 and 9 are the right and left elbow. The horizontal axes are time, in seconds.

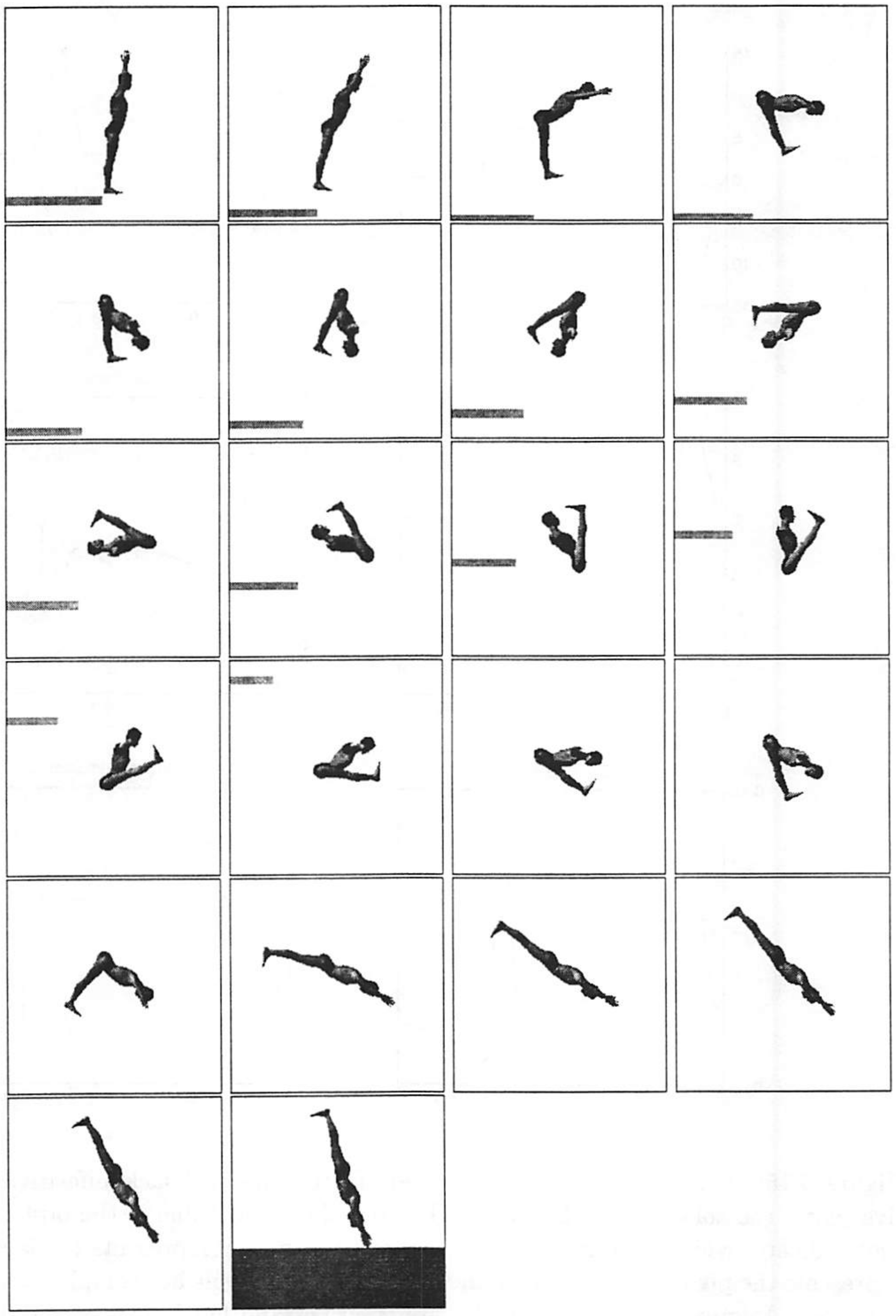


Figure 4.16: Frames from a simulation of the learned one-and-one-half-somersault pike dive.

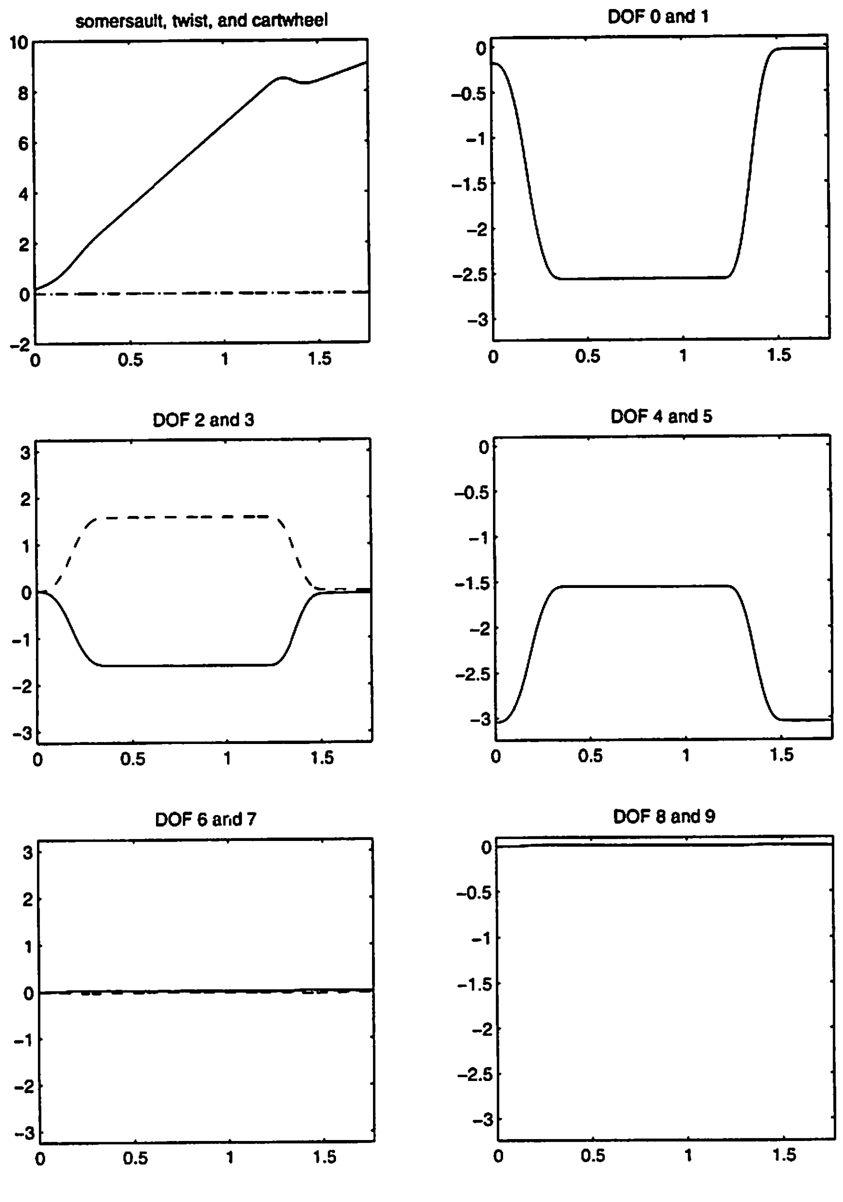


Figure 4.17: Body and joint angles (in radians) for the learned somersault dive. The first plot shows the somersault angle, the dashed line is the twist angle, and the dotted line is the cartwheel angle. For the other five plots, the solid line is the even DOF, and the dashed line is the odd DOF. Degrees of freedom 0 and 1 are the right and left hip, 2, 4, and 6 are the three Euler angles of the right shoulder, 3, 5, and 7 are the angles of the left shoulder, and 8 and 9 are the right and left elbow. The horizontal axes are time, in seconds.

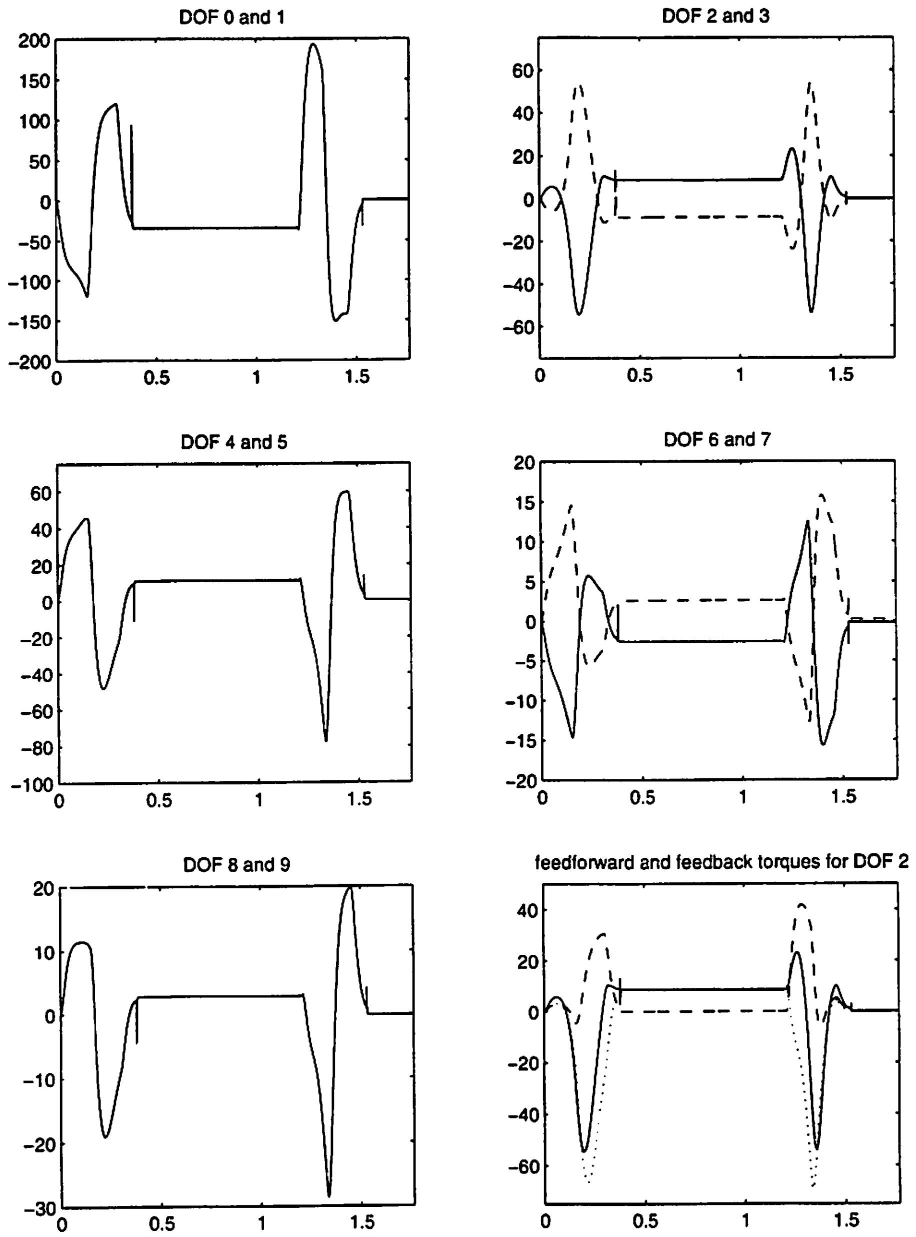


Figure 4.18: Torques (in Newton-meters) for the learned somersault dive. In the first five plots, the solid line is the even DOF, and the dashed line is the odd DOF. The last plot shows the total torque (solid), the feedforward torque (dashed), and the feedback torque (both during and between movements; dotted) for DOF 2. Degrees of freedom 0 and 1 are the right and left hip, 2, 4, and 6 are the three Euler angles of the right shoulder, 3, 5, and 7 are the angles of the left shoulder, and 8 and 9 are the right and left elbow. The horizontal axes are time, in seconds.

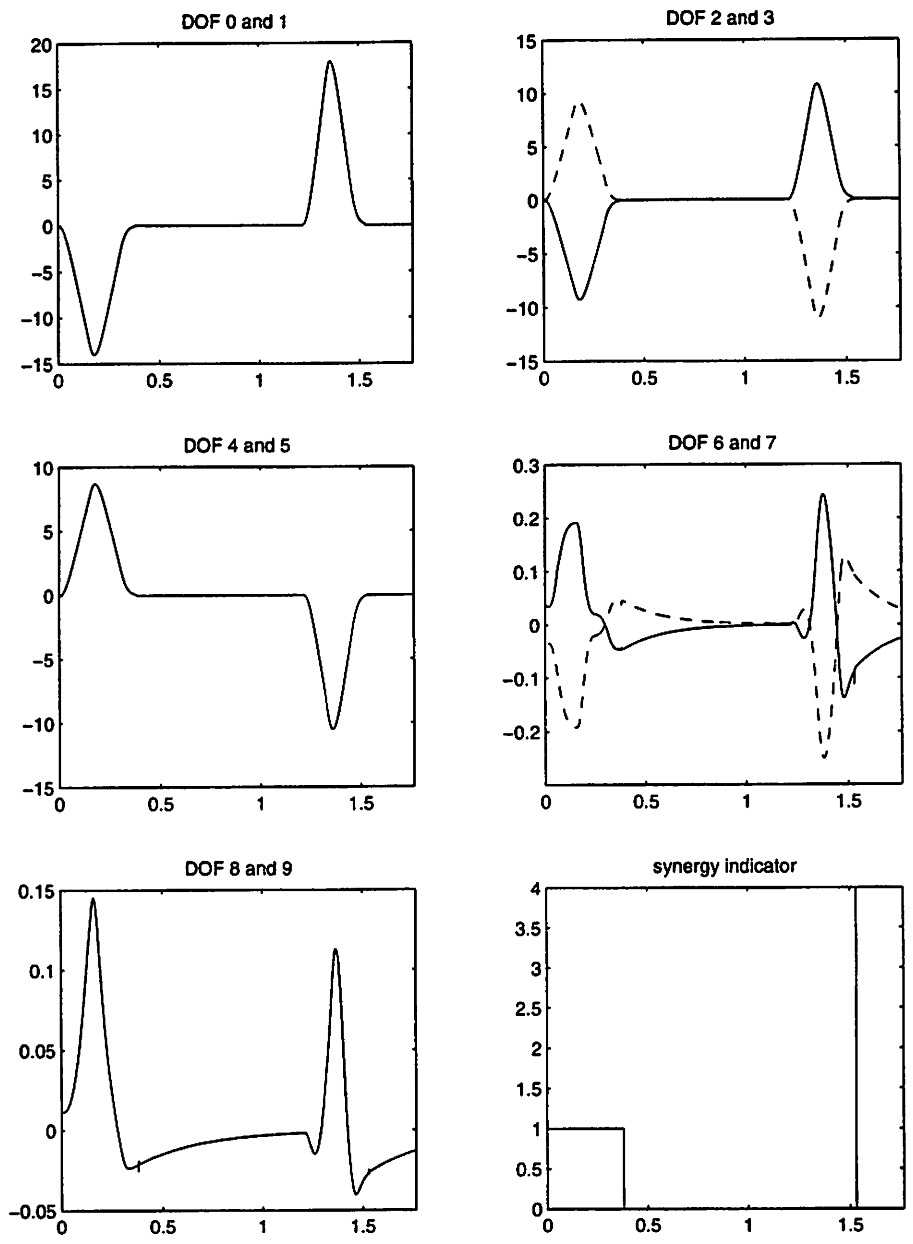
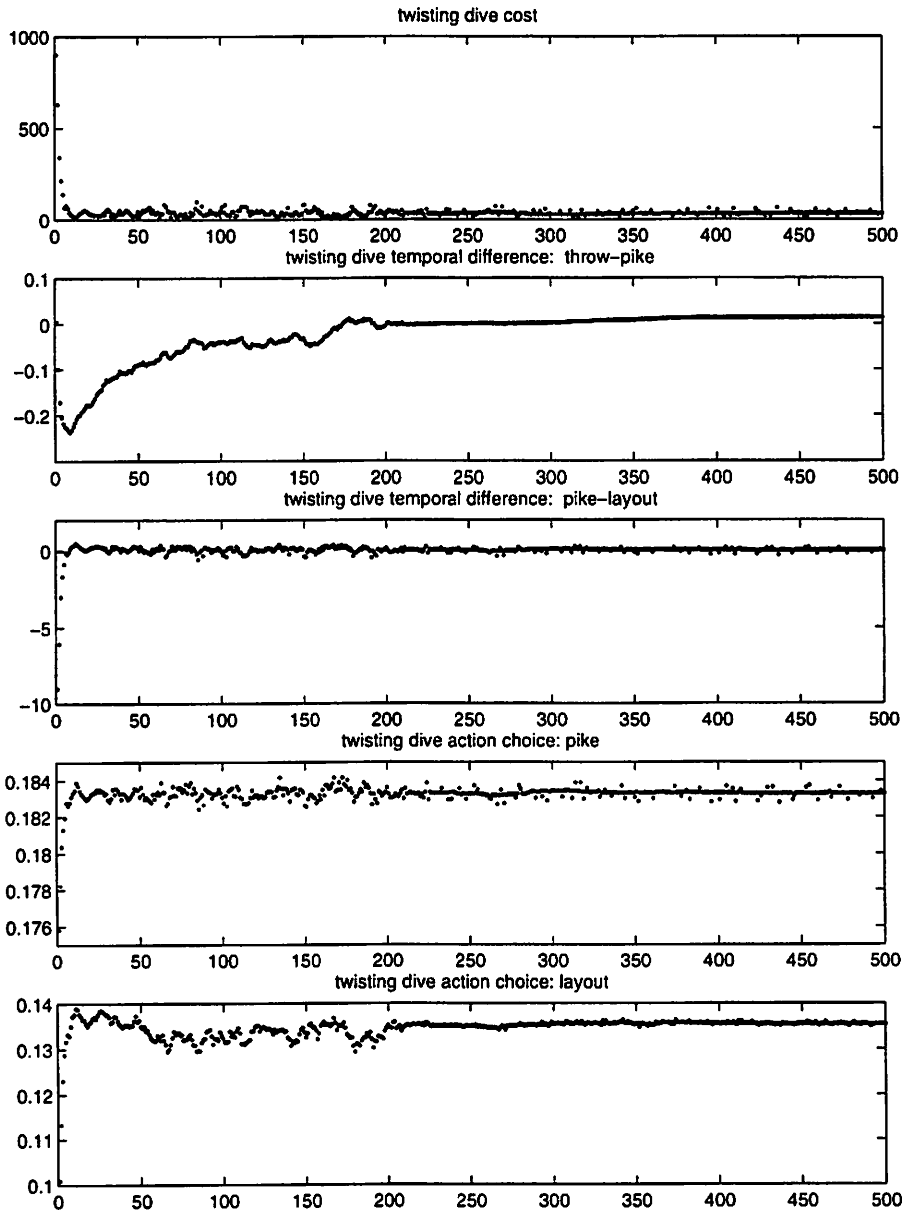


Figure 4.19: Velocities (in radians/second) for the learned somersault dive. In the first five plots, the solid line is the even DOF, and the dashed line is the odd DOF. The last plot indicates which synergy is active at any given time: 0 represents the layout synergy, 1 represents the pike synergy, and 4 indicates the state machine has completed all the required synergies. Degrees of freedom 0 and 1 are the right and left hip, 2, 4, and 6 are the three Euler angles of the right shoulder, 3, 5, and 7 are the angles of the left shoulder, and 8 and 9 are the right and left elbow. The horizontal axes are time, in seconds.


twisting dive cost
1000
500
0
0
50
100
150
200
250
300
350
400
450
500
twisting dive temporal difference: throw–pike
0.1
0
–0.1
–0.2
twisting dive temporal difference: pike–layout
0
–5
–10
twisting dive action choice: pike
0.184
0.182
0.18
0.178
0.176
twisting dive action choice: layout
0.14
0.13
0.12
0.11
0.1

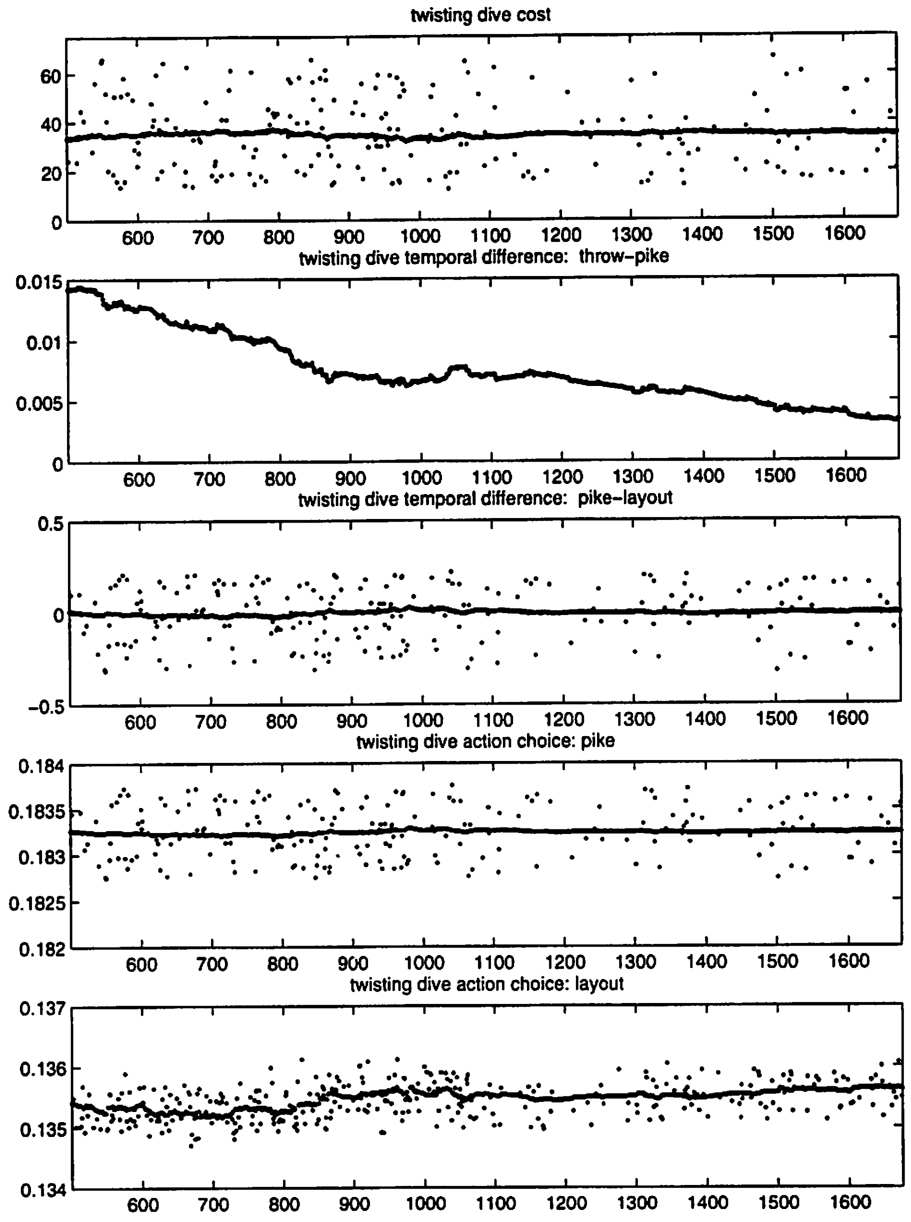


Figure 4.20: Training data for the twisting dive. The first set of plots shows the first 500 iterations, and the second set shows iterations 501 through 1675. Top plots: Dive cost, calculated as in Equation 4.47. Second and third: Temporal differences at the two different state transitions where learning occurs. Fourth and fifth: Scaled action choices (t_s, the time to wait before executing the indicated synergy) at the two different state transitions. The unscaled t_s values (times, in seconds) are two times the plotted scaled values.

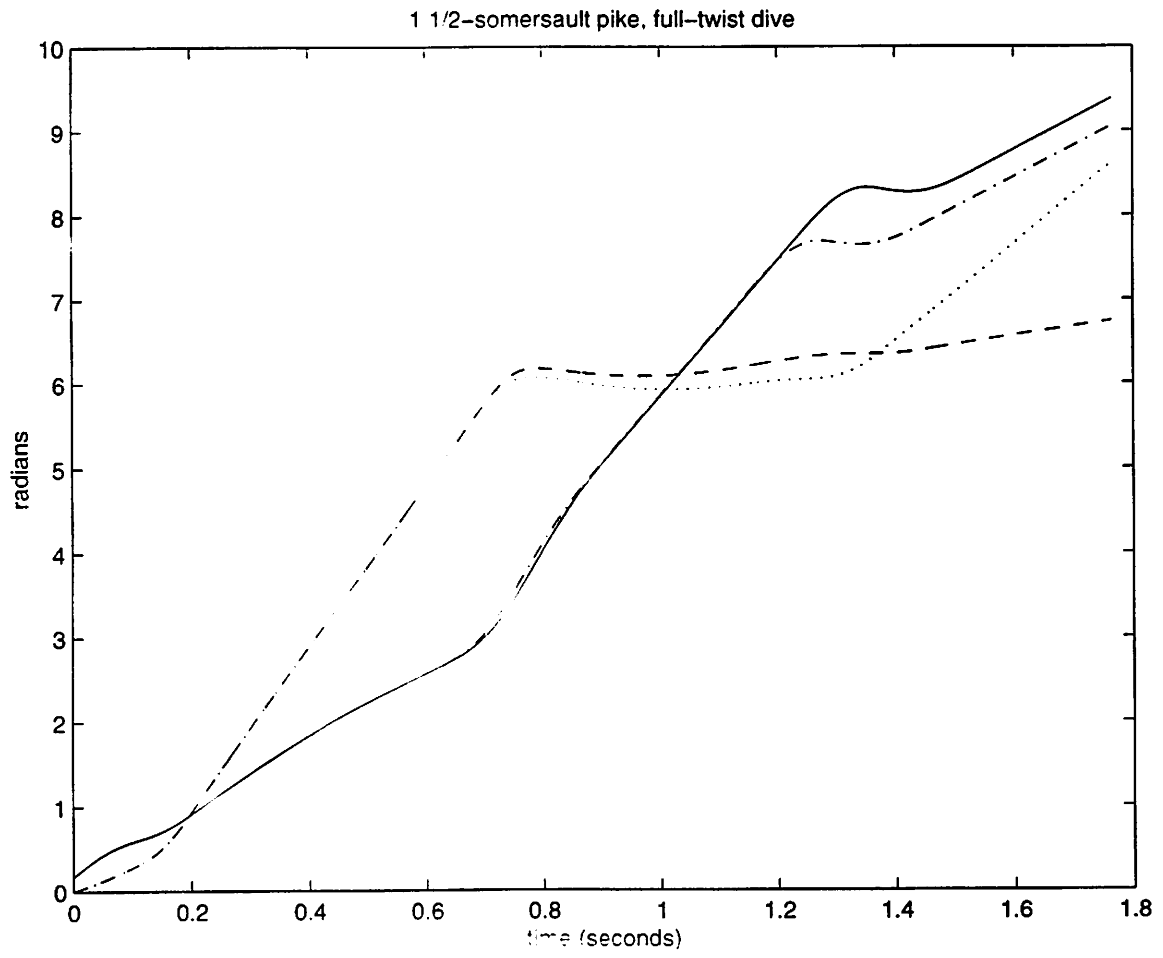


Figure 4.21: Comparison of the twisting dive after the "learning by watching" stage and after the full "learning by doing." The solid line indicates the somersault angle after the full learning, and the dashed line is the twist angle after the full learning; the dash-dot line is the somersault after the "learning by watching" phase, and the dotted line is the twist angle after "learning by watching." All angles are in radians.

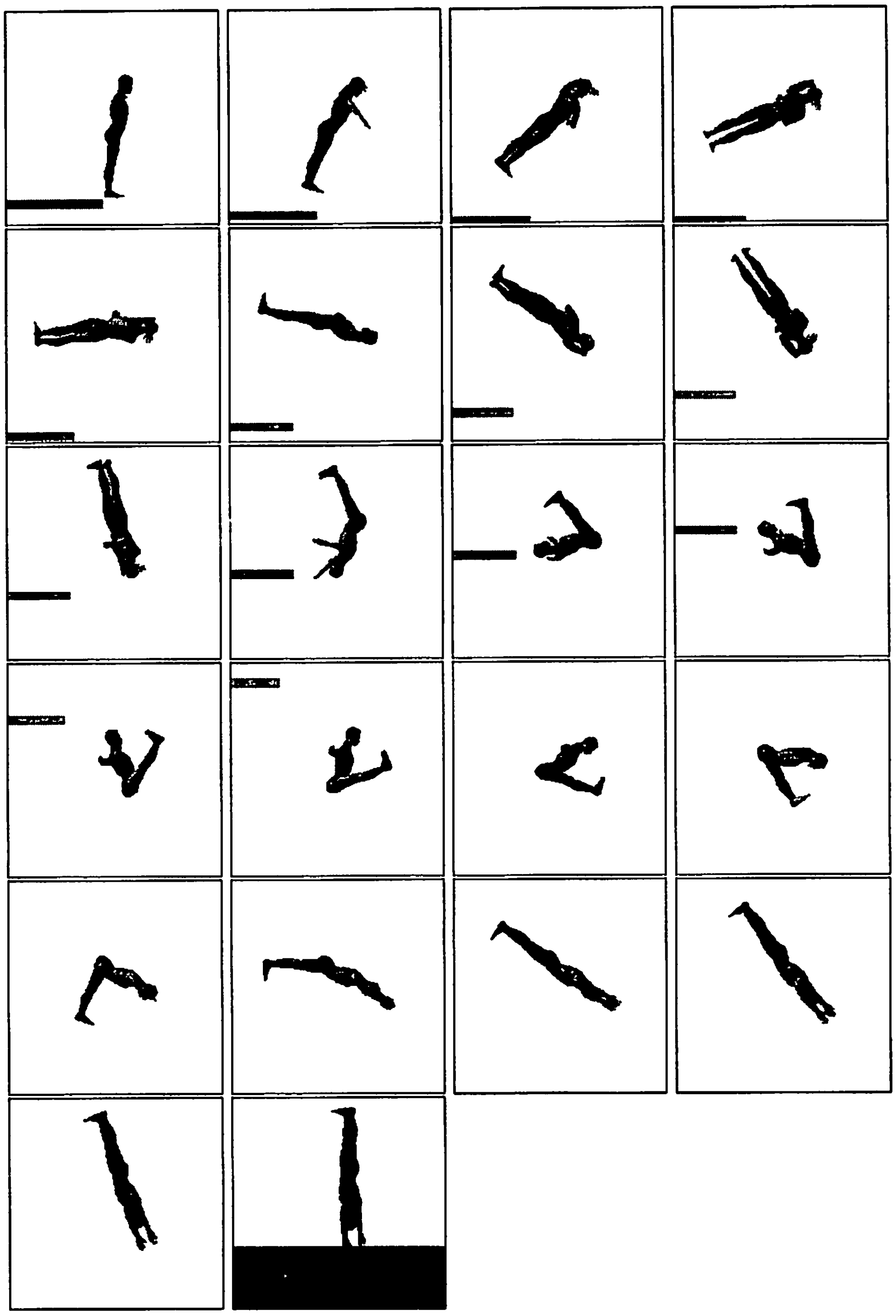


Figure 4.22: Frames from a simulation of the learned twisting dive.

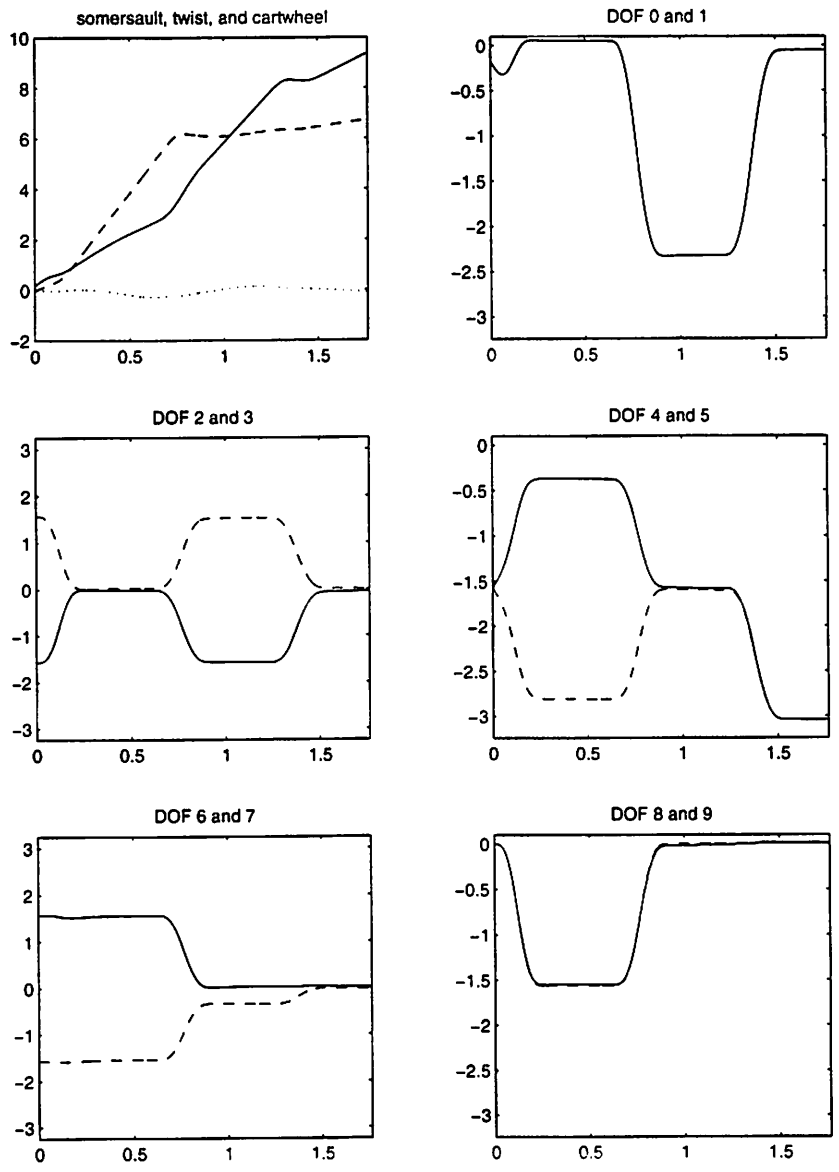


Figure 4.23: Body and joint angles (in radians) for the learned twisting dive. In the first plot, the solid line is the somersault angle, the dashed line is the twist angle, and the dotted line is the cartwheel angle. For the other five plots, the solid line is the even DOF, and the dashed line is the odd DOF. Degrees of freedom 0 and 1 are the right and left hip, 2, 4, and 6 are the three Euler angles of the right shoulder, 3, 5, and 7 are the angles of the left shoulder, and 8 and 9 are the right and left elbow. The horizontal axes are time, in seconds.

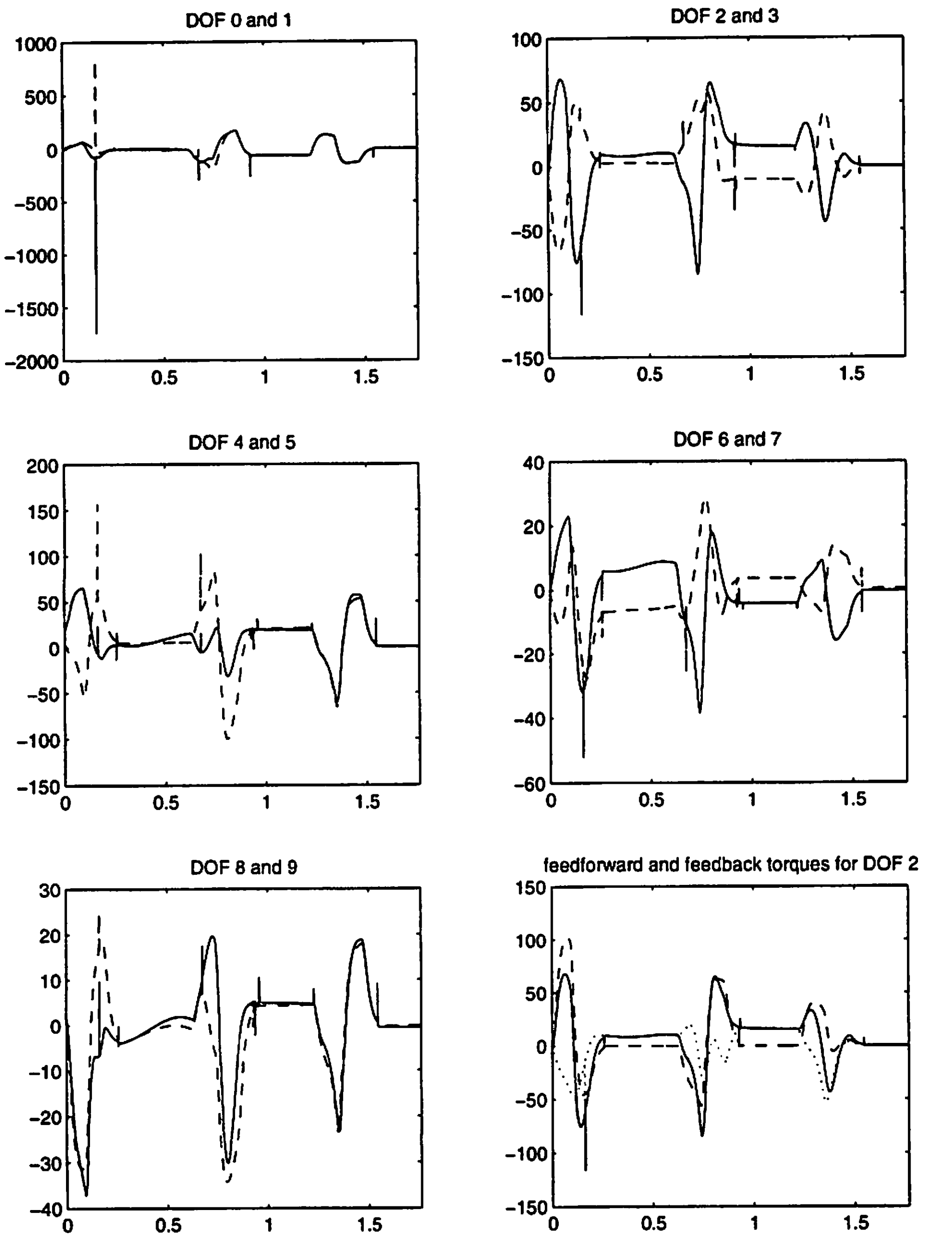


Figure 4.24: Torques (in Newton-meters) for the learned twisting dive. In the first five plots, the solid line is the even DOF, and the dashed line is the odd DOF. The large sharp spikes in the first plot are due to the joint limits (compare with Figure 4.23). The last plot shows the total torque (solid), the feedforward torque (dashed), and the feedback torque (both during and between movements; dotted) for DOF 2. Degrees of freedom 0 and 1 are the right and left hip, 2, 4, and 6 are the three Euler angles of the right shoulder, 3, 5, and 7 are the angles of the left shoulder, and 8 and 9 are the right and left elbow. The horizontal axes are time, in seconds.

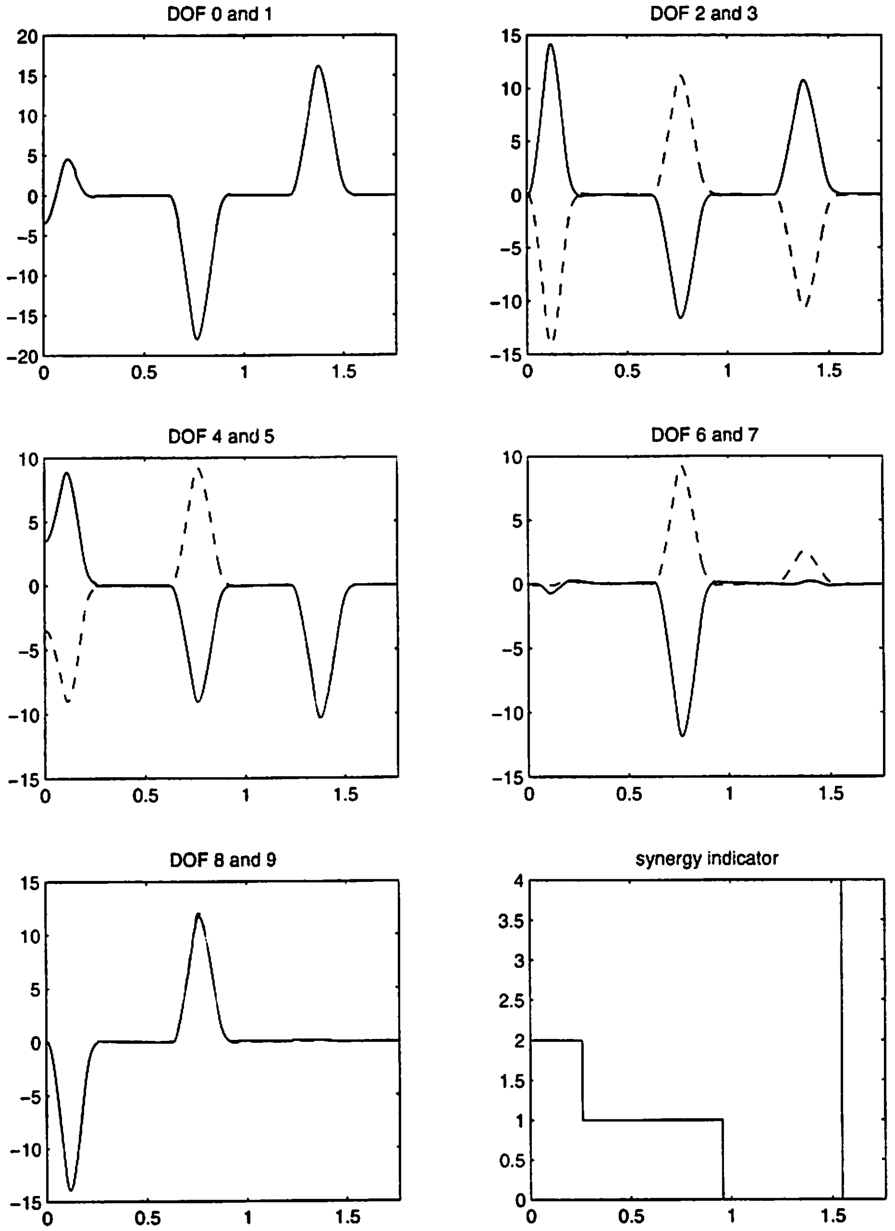


Figure 4.25: Velocities (in radians/second) for the learned twisting dive. In the first five plots, the solid line is the even DOF, and the dashed line is the odd DOF. The last plot indicates which synergy is active at any given time: 0 represents the layout synergy, 1 represents the pike synergy, 2 represents the throw synergy, and 4 indicates the state machine has completed all the required synergies. Degrees of freedom 0 and 1 are the right and left hip, 2, 4, and 6 are the three Euler angles of the right shoulder, 3, 5, and 7 are the angles of the left shoulder, and 8 and 9 are the right and left elbow. The horizontal axes are time, in seconds.

Chapter 5

Discussion

The results presented in Chapter 4 demonstrate that the learning controller presented in this dissertation is capable of learning to execute complex skills such as platform diving. In general, the approach presented here is good for learning complex skills which have a known goal, complicated or unknown dynamics, and known behavioral strategies for approaching the task (cognitive or linguistic information is available). There are difficulties with this approach, however, both in the representation and in the learning algorithms, and there is still a long way to go in addressing skill acquisition problems in general.

5.1 Behavioral Representation

The behavioral representation that was used for the diver, at the level of the multi-joint synergies ("throw," "pike," and "layout"), captured diving motions fairly well with only one variable parameter, t_s. There are, however, several drawbacks to this representation. First, there are several other parameters in the representation that need to be defined by hand in advance (see Table 4.3), such as the desired time T_d for each of the synergies and the desired joint excursions $\Delta\theta_d$ for each degree of freedom in each synergy. These could, of course, be added as variables in the action vector $\mathbf{v}$, but doing so would make the dive harder to learn. In addition, technique varied somewhat among the Olympic divers; some of them made some use of bending around the x-axis at the waist or pelvis, for example, which could be included in the behavioral representation with the addition of an extra degree of freedom.

A more serious drawback is that, in order for the system to be a Markov decision process, each behavior must end before the next one can be invoked; no blending of actions is possible. Also, the body position representation used in the controller design here, in which the configuration of the diver is simply represented by the binary string $\mathbf{p}$, renders the system no longer strictly Markov, since all positions classified as a particular $\mathbf{p}$ do not have exactly the same $\dot{\phi}$. Furthermore, human divers can do small, corrective motions that do not fit within any of the behavioral synergetic structures. Some of these motions involve more flexibility (of the back, primarily) than is possible with a rigid body model. In particular, the sensitivity to the time for leaving the throw position to enter the pike position seen in the simulation implies that human divers must use some kind of postural stabilization feedback, possibly with an anticipatory component, to ensure that the twist angle stays at 2π (that the angular velocity vector is realigned with the angular momentum vector). In addition to this stabilization, a human diver may have enhanced sensory feedback available that better allows the diver to anticipate the right time to come out of the throw. It may be possible to incorporate some of the repertoire of corrective movements and modifications of the multi-joint synergies that human divers use to correct errors in the twist angle or twist velocity (which should be zero after coming out of the throw) into the behavioral synergetic framework by adding new synergies or by adding parameters to existing ones. Any continuous, global feedback that may be used by divers would be difficult to incorporate into the Markov decision process structure, however, as well as possibly being difficult to formulate for a nonholonomic system without explicit path planning. Without such global feedback, high-level disturbance rejection during the dive is limited to that provided by the action selector, which takes into account the current state information vector $\mathbf{s_s} = [\phi_s, \phi_t, t, \mathbf{p}]$ through the coordinator network's approximation of Q, but which operates only at discrete times. There is, of course, some low-level disturbance rejection provided by the PD feedback. Global feedback would also be useful in refining the dive produced by the open-loop controller after "learning by doing" training, which may not quite reach the optimal control parameters, as mentioned in Section 4.2.3.

Another issue is how the student extracts the behavioral representation from linguistic information and from watching other performers during the "learning by watching" stage. The student must somehow decide which of the visual information describes the most essential features of the movement. Linguistic instructions from a coach must be translated into kinematic or dynamic constraints. For example, in the diver problem, the student knows that he or she must begin coming out of the twist at such a time as to end up with exactly 2π twist rotation. It is not obvious how to incorporate this foreknowledge into the behavioral or cost structure. In the control design presented here, the simulation does not learn the outcome of its timing choice until the end of the dive, and then this information must propagate back through the intervening states. A cost proportional to $(\phi_t - 2\pi)^2$ can be assigned at the end of the pike synergy, but doing so does not seem to improve learning, and the controller still has no implicit knowledge about the dynamics of the dive. A human knows in advance that, say, waiting for 1.5 seconds before coming out of the throw into the pike is always going to be too long, and will never try it; the controller, on the other hand, has no such cognitive understanding and must try everything allowed within the behavioral framework to know what choices will work. The cognitive understanding must therefore be built into the behavioral framework. The "learning by watching" behavioral stage alleviates this problem to some extent by initializing the $\tilde{Q}$ function and thus restricting the parameter search space.

5.2 Learning Techniques

Some of these drawbacks of the behavioral representation are intimately tied to the limitations of reinforcement learning techniques. By requiring the Markov decision process formulation, reinforcement learning restricts the behavioral representation to the sequential state machine model, with all the problems inherent in that model, as discussed above. Even with its drawbacks, the sequential state machine is a reasonable one for skills such as diving that involve a series of subroutines. For other skills that involve simultaneous, parallel subroutines, such as walking and running (see Section 2.5), bicycling, and windsurfing, however, a Markov decision process representation may not be as good a fit. In windsurfing, for example, the dynamics of the wind and water and the control actions employed by the surfer are much more naturally considered in continuous time than divided up into discrete time chunks, especially as a large component of the problem involves dealing with continuous, random disturbances. In bicycling, the parallel, coupled subroutines for balance, steering, and pedaling are also essentially continuous actions. In walking and running as well, although the three underlying subroutines can be implemented as a state machine (see [80], [79]) or viewed in the context of a return map, from a learning point of view, some of the subroutine parameters (possibly spring constants, for example, which are very important in locomotion) might be better learned with a continuous-time technique. As mentioned above, even in sequential skills like diving there can be overlap or blending between the serial control actions, as well as ongoing postural stabilization, so a more general, continuous-time learning model which still addresses the temporal credit assignment problem that arises when there are delayed rewards would be useful. Model predictive control, a method which has been widely used in process control for both continuous and discrete time system models (see [63], [83], [8]), bears a strong resemblance to reinforcement learning; an exploration of the connection between these two approaches could be an extremely fertile area for future research. A continuous-time reinforcement learning algorithm could also be useful in exploring other issues in learning, such as why progressing from feedback control to feedforward control during learning may be useful (see Section 2.5); in biological systems, using feedback at the start of learning when the open-loop control is still poor serves to protect the organism, but it may also perform other functions such as restricting the learning to a useful region of the state space.

Within the reinforcement learning paradigm, convergence results have been hard-won (see Section 4.2.2; [11]). For the diver problem, as mentioned in Section 4.2.3, because we do not have a model of the state transitions and costs, we are required to use Q-learning rather than, for example, TD(λ). Because the state space is partially continuous, we must use a function approximator rather than table lookup. These two factors, together with the fact that the diver system is only approximately Markov, complicate the picture in terms of convergence; there has as yet been no proof of convergence of a Q-like algorithm when combined with a function approximator, except for very special cases. With function approximators like neural networks that are nonlinear in their parameters, even TD(λ) may diverge; with Q-learning the situation is worse. In addition, nonlinear approximators are susceptible to local minimum problems.

With Q-learning, particularly in a deterministic system like the diver, exploration becomes a big issue as well. Exploration is required to ensure that the Q values of all potentially useful $(\mathbf{s}, \mathbf{v})$ pairs are being accurately modeled by the approximator; in a deterministic system, it is necessary to inject randomness into the choice of action to meet this requirement. The scope of the required randomness is limited by the "learning by watching" initialization, which eliminates extremely poor action choices from consideration, to some extent. Even with randomness, however, the Q-learning algorithm with a network approximator is prone to converge to a suboptimal choice of actions, as mentioned in Section 4.2.3. At each iteration, the algorithm improves the estimate $\tilde{Q}$ at the $(\mathbf{s}, \mathbf{v})$ point just visited. A neural network is a global approximator, so this adjustment affects the $\tilde{Q}$

estimate at all other points as well, and thus moves the minimum of the $\tilde{Q}$ function. A small weight adjustment can have quite a large effect on the location of the $\tilde{Q}$ minimum and thus on the greedy action. Even if small learning rates are used, updating the $\tilde{Q}$ value at one point distorts the shape of the approximation at neighboring points to some extent, and the network tends to "forget" what it has learned about those points previously. Once the temporal differences become small, the algorithm tends to get stuck at its current estimate of the minimum, even though random action choices force the controller to sample nearby points occasionally. Adding the small random component η to the weight update improves matters somewhat, but the algorithm still tends not to reach the lowest possible dive error. This problem is similar to the requirement of persistent excitation in adaptive control, but is exacerbated by the global nature of the network approximation. It is possible that using much larger networks, perhaps with steeper activation functions, would help this problem by allowing more neurons to contribute effectively to the $\tilde{Q}$ estimate in each region. Also, with a mechanism for global feedback, this limitation of the open-loop learning would be less of a problem, as mentioned in Section 5.1.

The $\tilde{Q}$ minimum, and thus the greedy action, do not necessarily travel toward the true minimum at each step in the algorithm. The shape of the "learning by watching" $\tilde{Q}$ initialization is thus quite important. Depending on its shape, the action choices may, at the start of the "learning by doing" phase, head toward or away from the true optimal values. If they head away from the optimum very much, the coordinator network may end up in a suboptimal local minimum. In addition, since the $\tilde{Q}$ minimum tends to stop moving when the temporal differences get small, better final dive costs are usually obtained if the estimated minimum starts out moving in the right direction. "Learning by doing" for the controller is therefore different from human learning in that in order to start the Q-learning algorithm near the best local minimum and moving in the right direction, fairly accurate "learning by watching" initialization is required. The human learning algorithm is obviously much more sophisticated and is able to make use of a higher level of understanding; it can therefore succeed with much rougher initial timing estimates.

5.3 Conclusion

Though there are several hurdles that need to be overcome in applying reinforcement learning techniques to continuous, nonlinear problems like the diver, the approach is promising. By using a behavioral control structure to simplify the control representation, the learning problem is reduced to the relatively simple task of learning a set of parameters at each hierarchical level. Thus, complex skills which are difficult or impossible to control using conventional methods become feasible: the controller described in this dissertation successfully learned to execute three different platform dives.

The ability of a controller to learn complex, natural skills has applications in dynamic computer animation as well as robotics. In addition, the idea of a behavioral structure together with learning algorithms can be used in an identification context, such as for visual learning or gesture recognition. Eventually, a deeper understanding of machine and robotic skill acquisition could lead to insights into human motor learning that would be useful in sports training or rehabilitation contexts as well as theoretically.

Appendix A

Inertial Compensator

The calculations for the inertial compensator are based on the product of exponentials formulation for kinematics; see [70] for a full exposition. Each degree of freedom can be described by a twist $\xi \in \mathbb{R}^6$. The twist can be written

$$\xi = \begin{bmatrix} \mathbf{v} \\ \omega \end{bmatrix}, \tag{A.1}$$

where ω is the axis of rotation of the degree of freedom and $\mathbf{v} = -\omega \times \mathbf{q}$, with $\mathbf{q}$ a point on the rotation axis. ξ can be converted into an element of se(3) through the $\hat{}$ (cross product) operator:

$$\hat{\xi} = \begin{bmatrix} \hat{\omega} & \mathbf{v} \\ 0 & 0 \end{bmatrix} \tag{A.2}$$

where $\hat{\omega} \in so(3)$:

$$\hat{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{A.3}$$

As mentioned in Section 4.1.1, the inertial compensator is greatly simplified by considering each limb independently. For one arm, then, there are four degrees of freedom: three at the shoulder and one at the elbow (degrees of freedom 2, 4, 6, and 8 for the right arm and 3, 5, 7, and 9 for the left). For the following discussion, these degrees of freedom will be numbered one through four. The corresponding twists are:

$$\xi_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \ \xi_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \ \xi_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \ \xi_4 = \begin{bmatrix} -z_j \\ 0 \\ x_j \\ 0 \\ 1 \\ 0 \end{bmatrix}, \tag{A.4}$$

where $\begin{bmatrix} x_j \\ y_j \\ z_j \end{bmatrix}$ are the coordinates of the elbow joint relative to the shoulder joint in the zero position. For the ith degree of freedom, the body frame Jacobian is given by

$$\mathbf{J}_i = \left[\xi_1^\dagger \cdots \xi_i^\dagger \ 0 \cdots 0\right] \tag{A.5}$$

$\xi_j^\dagger$, the instantaneous twist of the jth degree of freedom relative to the ith link, is

$$\xi_j^\dagger = \mathrm{Ad}^{-1}_{e^{\hat{\xi}_j\theta_j}\cdots e^{\hat{\xi}_i\theta_i}\mathbf{g}_i(0)} \xi_j, \tag{A.6}$$

where θ_j is the angle of the jth degree of freedom and $\mathbf{g}_i(0)$ is an element of SE(3) describing the position and orientation of the link corresponding to joint i in the zero configuration. The inverse adjoint operator corresponding to an arbitrary element $\mathbf{g}$ of SE(3), in homogeneous coordinates,

$$\mathbf{g} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ 0 & 1 \end{bmatrix}, \tag{A.7}$$

with $\mathbf{R} \in SO(3)$ and $\mathbf{p} \in \mathbb{R}^3$, is given by

$$\mathrm{Ad}_{\mathbf{g}}^{-1} = \begin{bmatrix} \mathbf{R}^{\mathrm{T}} & -\mathbf{R}^{\mathrm{T}}\hat{\mathbf{p}} \\ 0 & \mathbf{R}^{\mathrm{T}} \end{bmatrix} \tag{A.8}$$

The exponential of a twist can be obtained as

$$e^{\hat{\xi}\theta} = \begin{bmatrix} e^{\hat{\omega}\theta} & (\mathbf{I} - e^{\hat{\omega}\theta})(\omega \times \mathbf{v}) + \omega\omega^T\mathbf{v}\theta \\ 0 & 1 \end{bmatrix}, \tag{A.9}$$

where $\mathbf{I}$ is the identity in $\mathbb{R}^3$ and $e^{\hat{\omega}\theta}$ is the rotation matrix $\mathbf{R} \in SO(3)$, which can be obtained with Rodrigues' formula:

$$e^{\hat{\omega}\theta} = \mathbf{I} + \hat{\omega}\sin\theta + \hat{\omega}^2(1-\cos\theta) \tag{A.10}$$

Finally, the mass matrix for the limb is given by

$$\mathbf{M}(\theta) = \sum_{i=1}^{n} \mathbf{J}_i^{\mathrm{T}}(\theta)\mathcal{M}_i\mathbf{J}_i(\theta), \tag{A.11}$$

where n is the number of degrees of freedom in the limb, $\mathcal{M}_i$ is the generalized inertia matrix for the link corresponding to the ith degree of freedom, and θ is a vector of the angles of all the degrees of freedom. $\mathcal{M}_i$ takes the form

$$\mathcal{M}_i = \begin{bmatrix} m_i\mathbf{I} & 0 \\ 0 & \mathcal{I}_i \end{bmatrix}, \tag{A.12}$$

where m_i is the mass of the link corresponding to the ith degree of freedom and $\mathcal{I}_i$ is the inertia matrix for that link. If the principal axes of the link are aligned with the global axes in the zero position, this matrix is diagonal.

For the arms, there are four degrees of freedom: three Euler angles in the shoulder and elbow flexion. $\mathcal{M}_1 = \mathcal{M}_2 = 0$, since the first three degrees of freedom are all part of the same joint; the first two "links" are virtual. Using the equations above, tedious calculation obtains:

$$\mathbf{J}_3 = \begin{bmatrix} -c_2y_u + s_2s_3z_u & c_3z_u & -y_u & 0 \\ c_2x_u + s_2c_3z_u & -s_3z_u & x_u & 0 \\ -s_2s_3x_u - s_2c_3y_u & s_3y_u - c_3x_u & 0 & 0 \\ -s_2c_3 & s_3 & 0 & 0 \\ s_2s_3 & c_3 & 0 & 0 \\ c_2 & 0 & 1 & 0 \end{bmatrix} \tag{A.13}$$

$$\mathbf{J}_4 = \begin{bmatrix} \begin{array}{l} (s_2c_3s_4 - c_2c_4)y_l \\ +s_2s_3z_l \\ +s_2s_3s_4x_j \\ -s_2s_3(1-c_4)z_j \end{array} & \begin{array}{l} -s_3s_4y_l + c_3z_l \\ +c_3s_4x_j \\ -c_3(1-c_4)z_j \end{array} & -c_4y_l & z_l - z_j \\ \begin{array}{l} (c_2c_4 - s_2c_3s_4)x_l \\ +(c_2s_4 + s_2c_3c_4)z_l \\ +(c_2(1-c_4) + s_2c_3s_4)x_j \\ +(s_2c_3(1-c_4) - c_2s_4)z_j \end{array} & \begin{array}{l} s_3s_4x_l - s_3c_4z_l \\ -s_3s_4x_j \\ -s_3(1-c_4)z_j \end{array} & \begin{array}{l} c_4x_l + s_4z_l \\ +x_j(1-c_4) \\ -s_4z_j \end{array} & 0 \\ \begin{array}{l} -s_2s_3x_l \\ -(s_2c_3c_4 + c_2s_4)y_l \\ +s_2s_3(1-c_4)x_j \\ +s_2s_3s_4z_j \end{array} & \begin{array}{l} -c_3x_l + s_3c_4y_l \\ +c_3(1-c_4)x_j \\ +c_3s_4z_j \end{array} & -s_4y_l & -x_l + x_j \\ -s_2c_3c_4 - c_2s_4 & s_3c_4 & -s_4 & 0 \\ s_2s_3 & c_3 & 0 & 1 \\ -s_2c_3s_4 + c_2c_4 & s_3s_4 & c_4 & 0 \end{bmatrix} \tag{A.14}$$

where $\begin{bmatrix} x_u \\ y_u \\ z_u \end{bmatrix}$ and $\begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix}$ are the coordinates of the center of mass of the upper and lower arm (plus hand), respectively, in the zero configuration, $\begin{bmatrix} x_j \\ y_j \\ z_j \end{bmatrix}$ are, as above, the coordinates of the elbow joint in the zero configuration, and c_i and s_i are the cosine and sine, respectively, of the angle of the ith degree of freedom. All coordinates are with respect to the shoulder joint.

Making use of the fact that

$$\mathbf{J}_4^{\mathrm{T}}(\theta)\mathcal{M}_4\mathbf{J}_4(\theta) = \mathbf{J}_{4\mathrm{l}}^{\mathrm{T}}(\theta)\mathcal{M}_{4\mathrm{l}}\mathbf{J}_{4\mathrm{l}}(\theta) + \mathbf{J}_{4\mathrm{h}}^{\mathrm{T}}(\theta)\mathcal{M}_{4\mathrm{h}}\mathbf{J}_{4\mathrm{h}}(\theta), \tag{A.15}$$

where the subscript l indicates the lower arm and the subscript h indicates the hand, we can simplify calculation by making use of the diagonality of the inertia matrices in the human model. Equation (A.11) then becomes:

$$\mathbf{M} = \mathbf{J}_3^{\mathrm{T}}(\theta)\mathcal{M}_3\mathbf{J}_3(\theta) + \mathbf{J}_{4\mathrm{l}}^{\mathrm{T}}(\theta)\mathcal{M}_{4\mathrm{l}}\mathbf{J}_{4\mathrm{l}}(\theta) + \mathbf{J}_{4\mathrm{h}}^{\mathrm{T}}(\theta)\mathcal{M}_{4\mathrm{h}}\mathbf{J}_{4\mathrm{h}}(\theta) \tag{A.16}$$

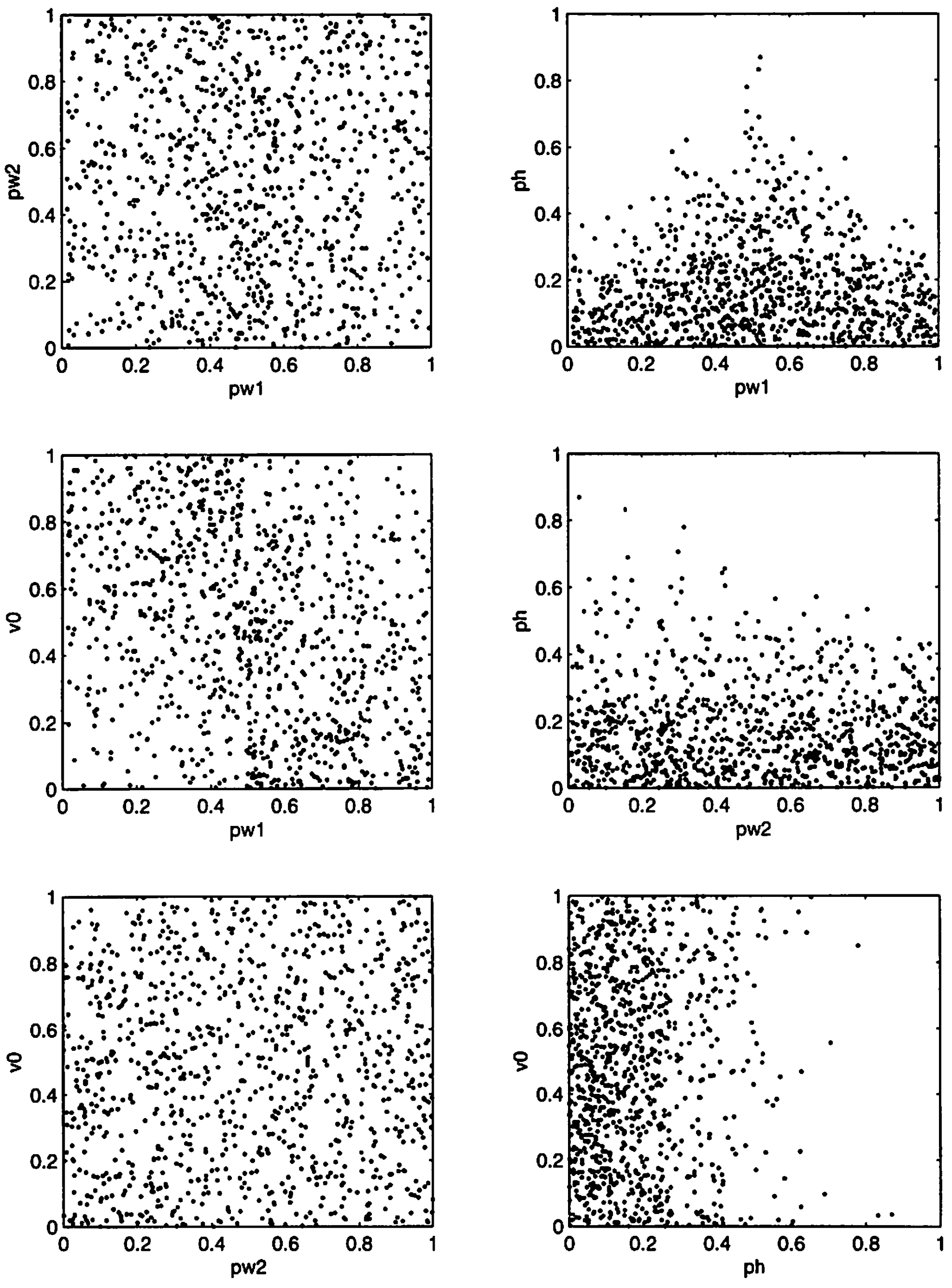
$\mathcal{M}_{4\mathrm{l}}$ and $\mathcal{M}_{4\mathrm{h}}$ are written with respect to the lower arm and hand separately, and $\mathbf{J}_{4\mathrm{l}}(\theta)$ and $\mathbf{J}_{4\mathrm{h}}(\theta)$ can similarly be obtained from Equation (A.14) using the coordinates of the lower arm and hand individually.

For the leg limbs, no calculation is needed, as there is only one degree of freedom involved, hip flexion. The mass matrices are therefore constant.

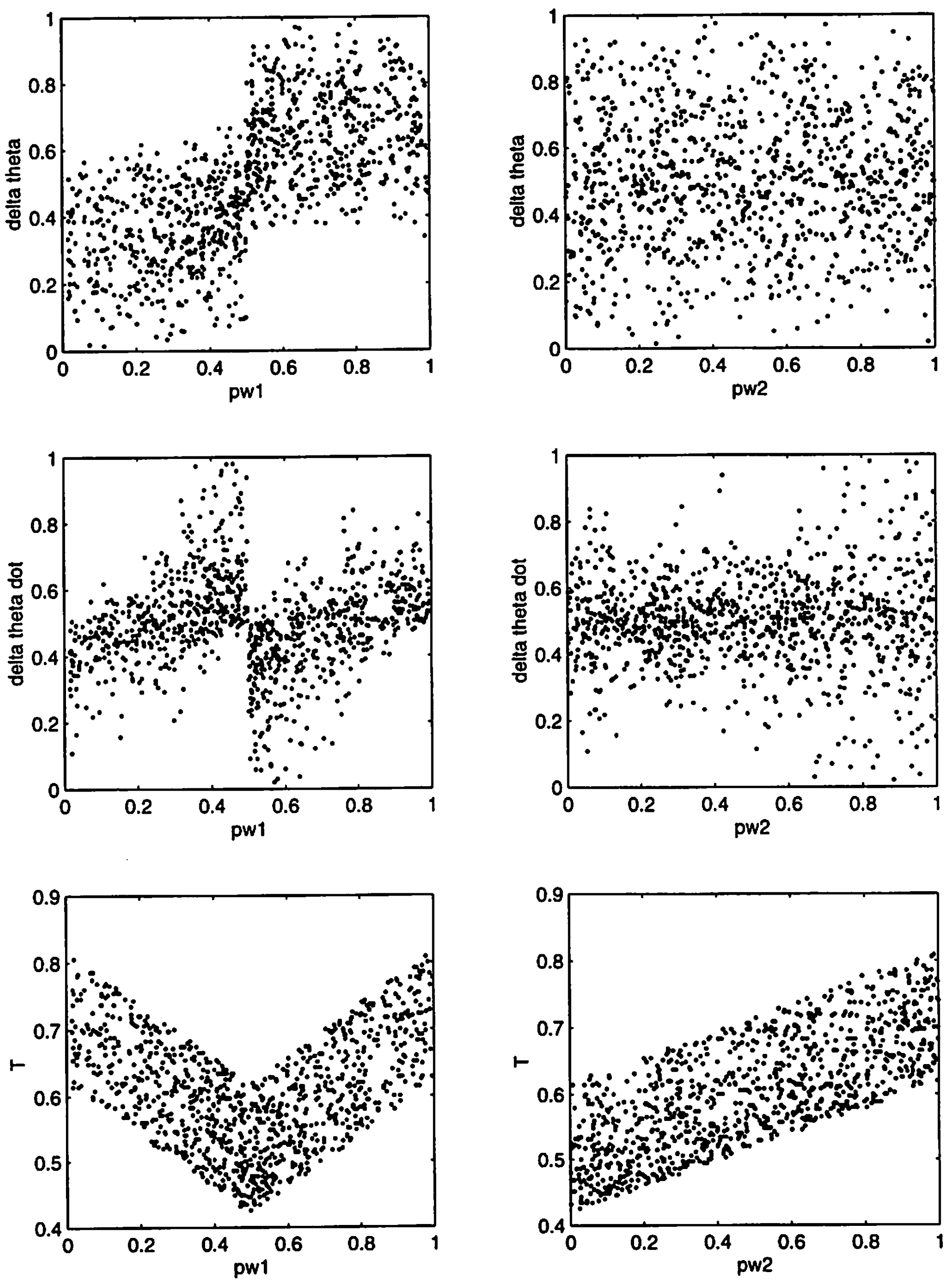
Appendix B

Data Plots

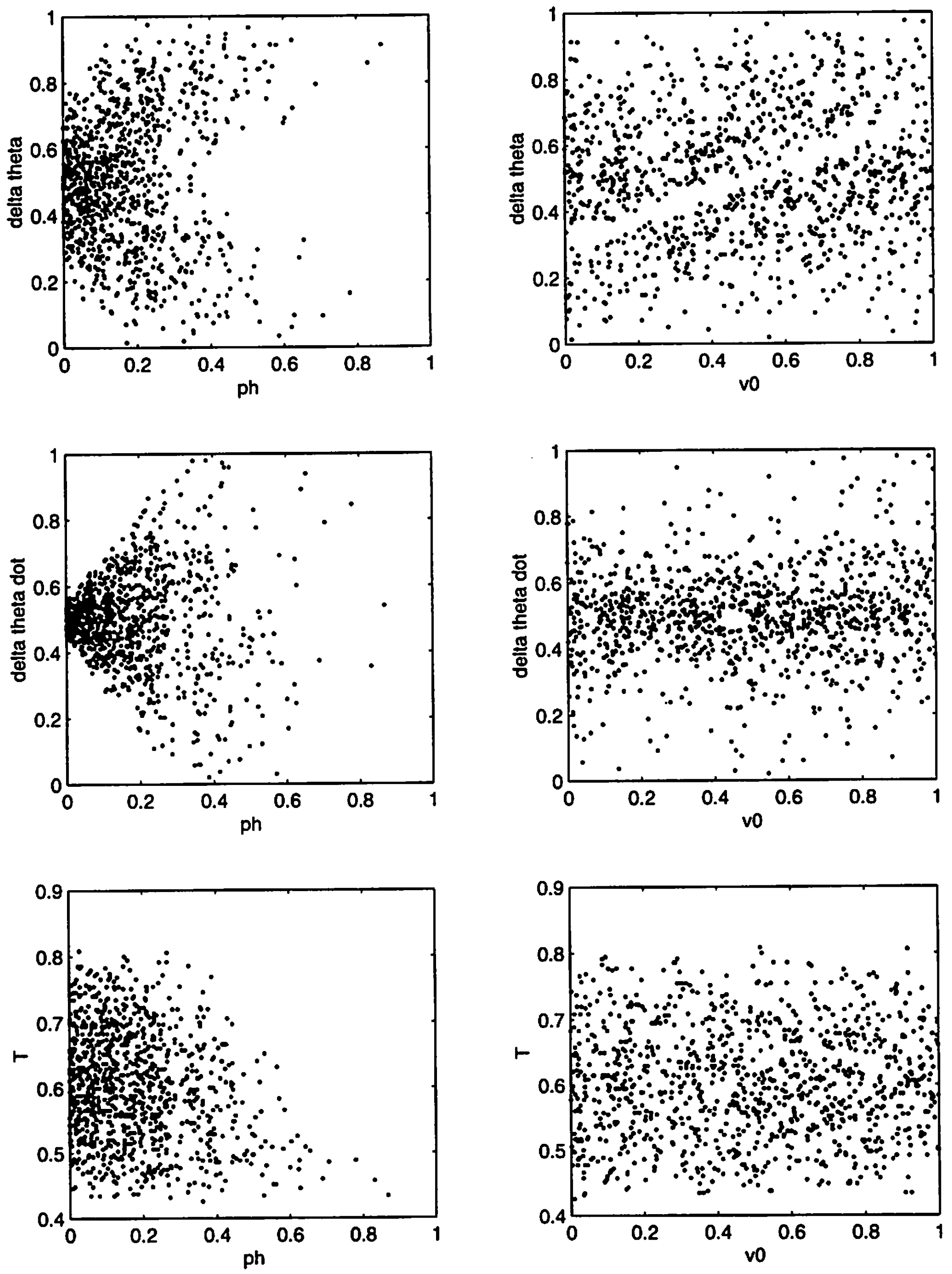
A



B


delta theta
pw1
delta theta
pw2
delta theta dot
pw1
delta theta dot
pw2
T
pw1
T
pw2
0
0.2
0.4
0.6
0.8
1
0.5
0.7
0.9

C


delta theta
ph
delta theta
v0
delta theta dot
ph
delta theta dot
v0
T
ph
T
v0

D

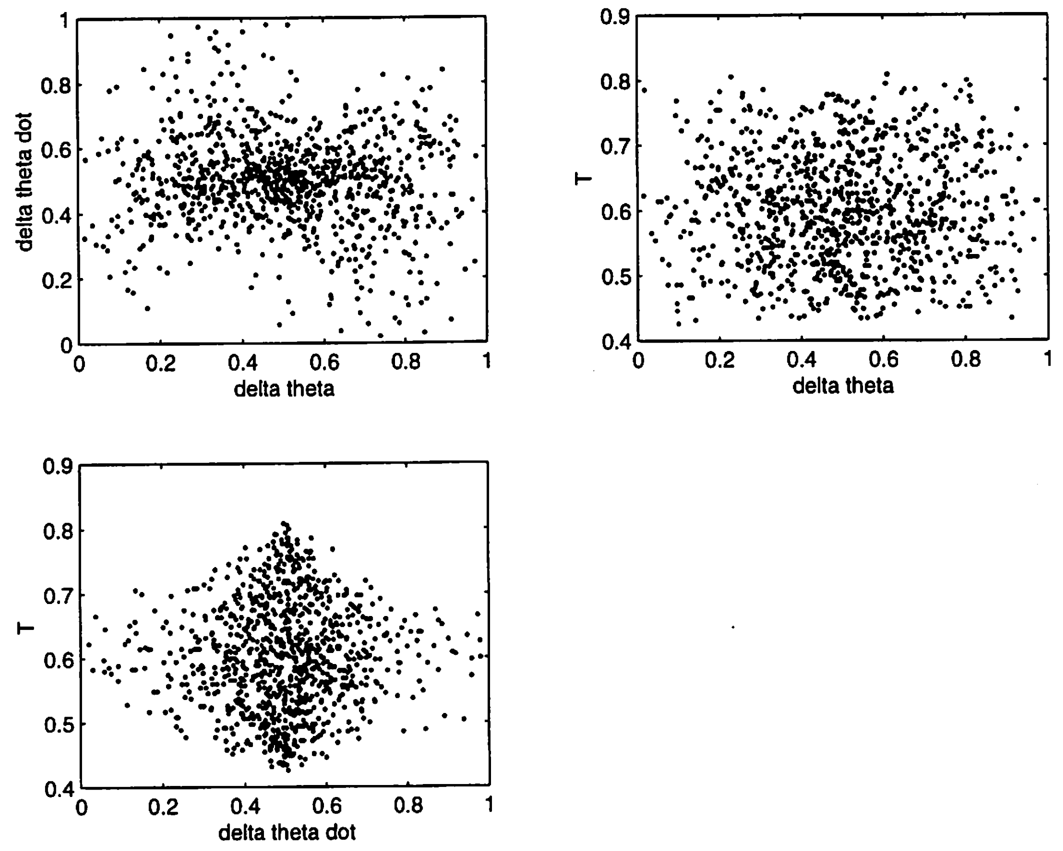
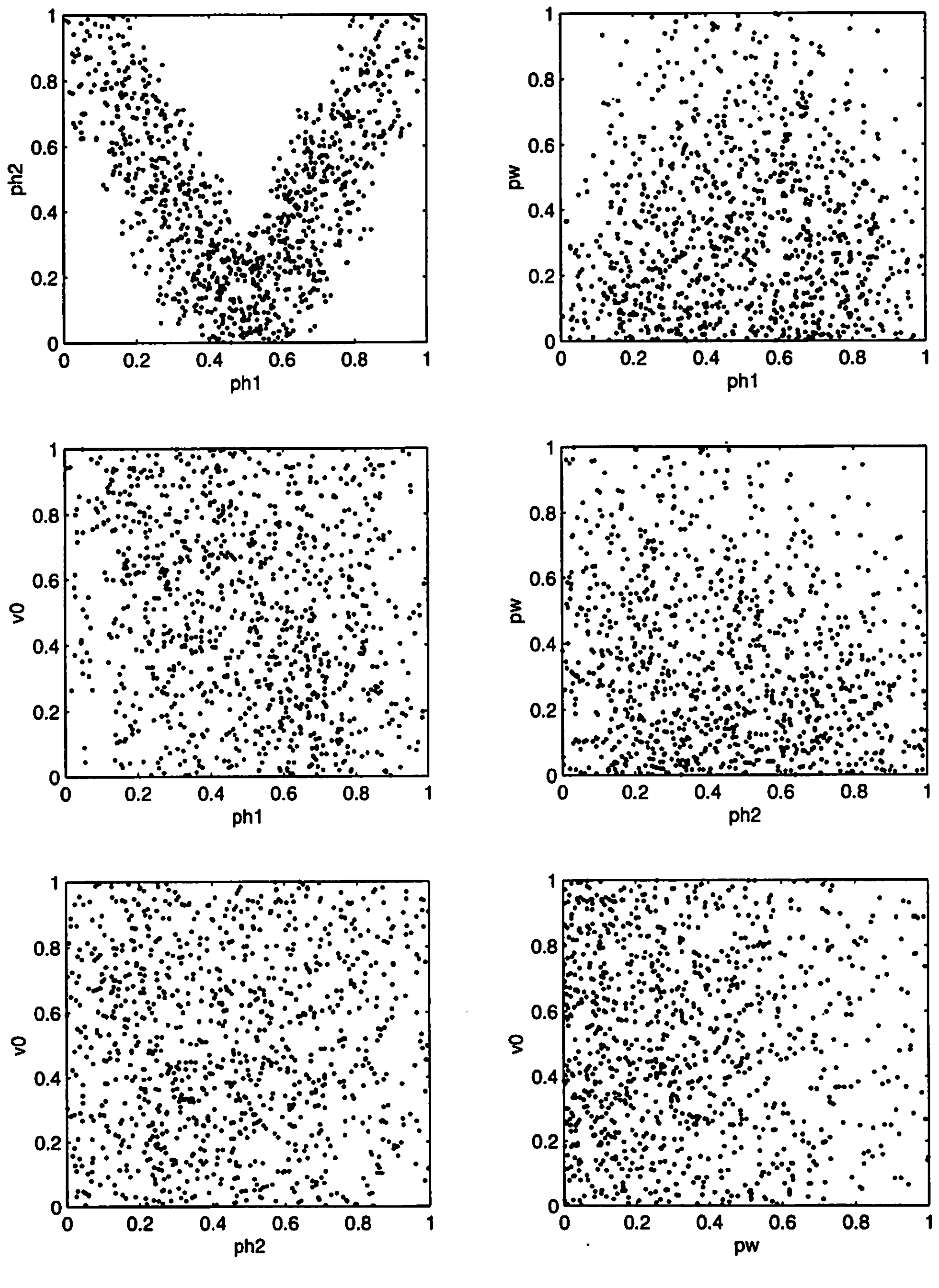
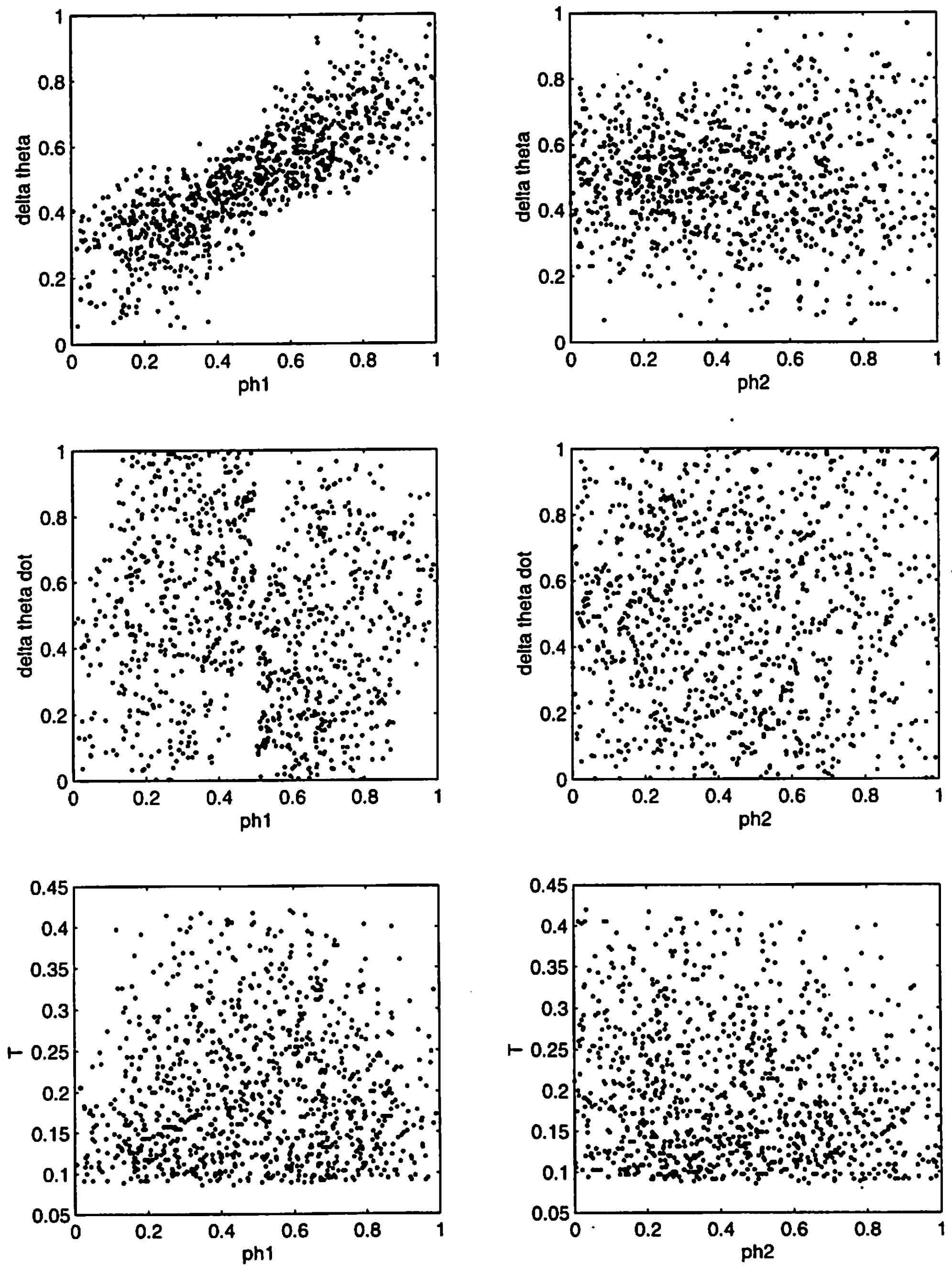


Figure B.1: Plots of the 1000-point data set for DOF 2, PWM mode. A. Control parameters and sensor input v_0 plotted against each other. These values were generated randomly; portions of the space that do not appear in the data set cause the simulation to run into the joint limits or to exceed the prespecified limits on $\Delta\dot{\theta}$ (see Section 4.1.2). B. Control parameters pw1 and pw2 plotted against the y values $\Delta\theta$, $\Delta\dot{\theta}$, and T. C. Control parameter ph and sensor input v_0 plotted against the y values. D. y values plotted against each other.

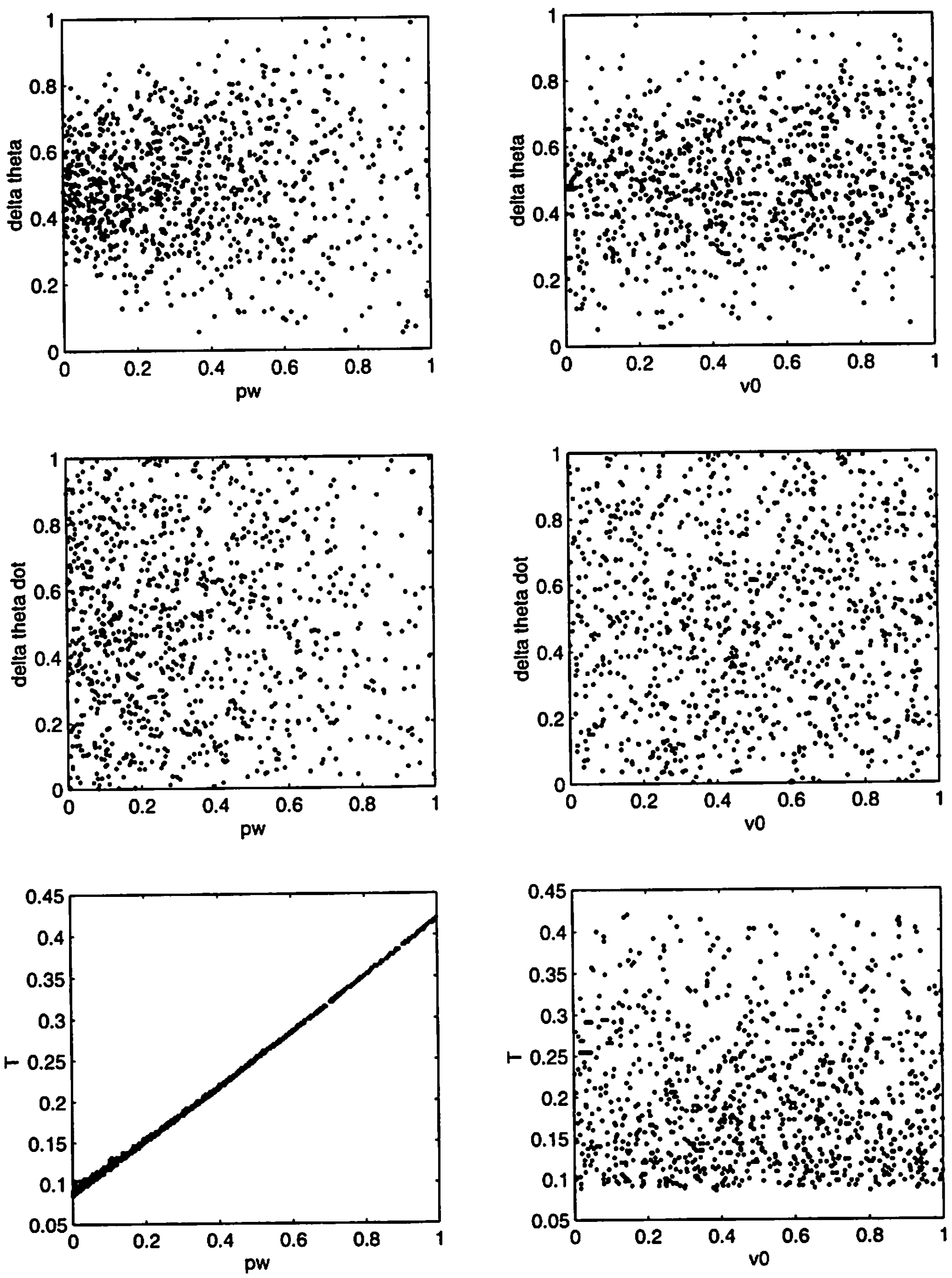
A



B


delta theta
ph1
delta theta
ph2
delta theta dot
ph1
delta theta dot
ph2
T
ph1
T
ph2

C


delta theta
pw
v0
delta theta dot
T
0
0.2
0.4
0.6
0.8
1
0.05
0.1
0.15
0.25
0.3
0.35
0.45

D

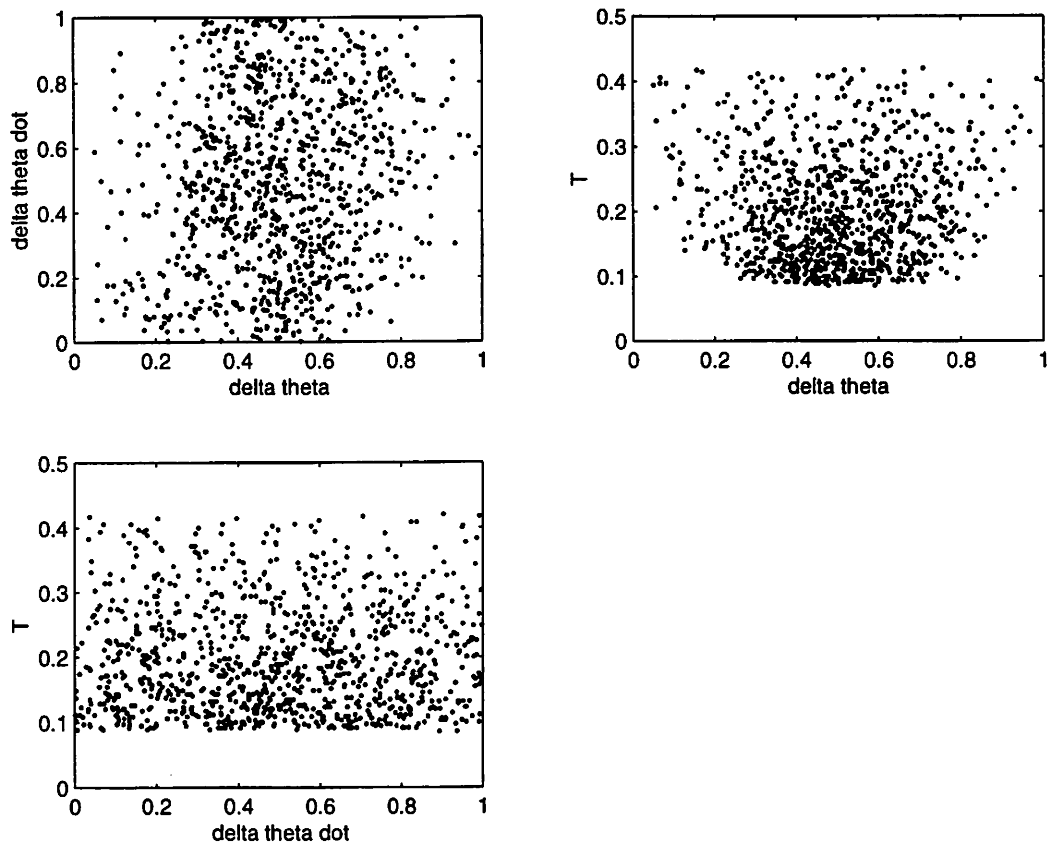


Figure B.2: Plots of the 1000-point data set for DOF 2, PHM mode. A. Control parameters and sensor input v_0 plotted against each other. These values were generated randomly; portions of the space that do not appear in the data set cause the simulation to run into the joint limits or to exceed the prespecified limits on $\Delta\dot{\theta}$ (see Section 4.1.2). B. Control parameters ph1 and ph2 plotted against the **y** values $\Delta\theta$, $\Delta\dot{\theta}$, and T. C. Control parameter pw and sensor input v_0 plotted against the **y** values. D. **y** values plotted against each other.

Appendix C

Single-DOF Controllers: Sample Training Results

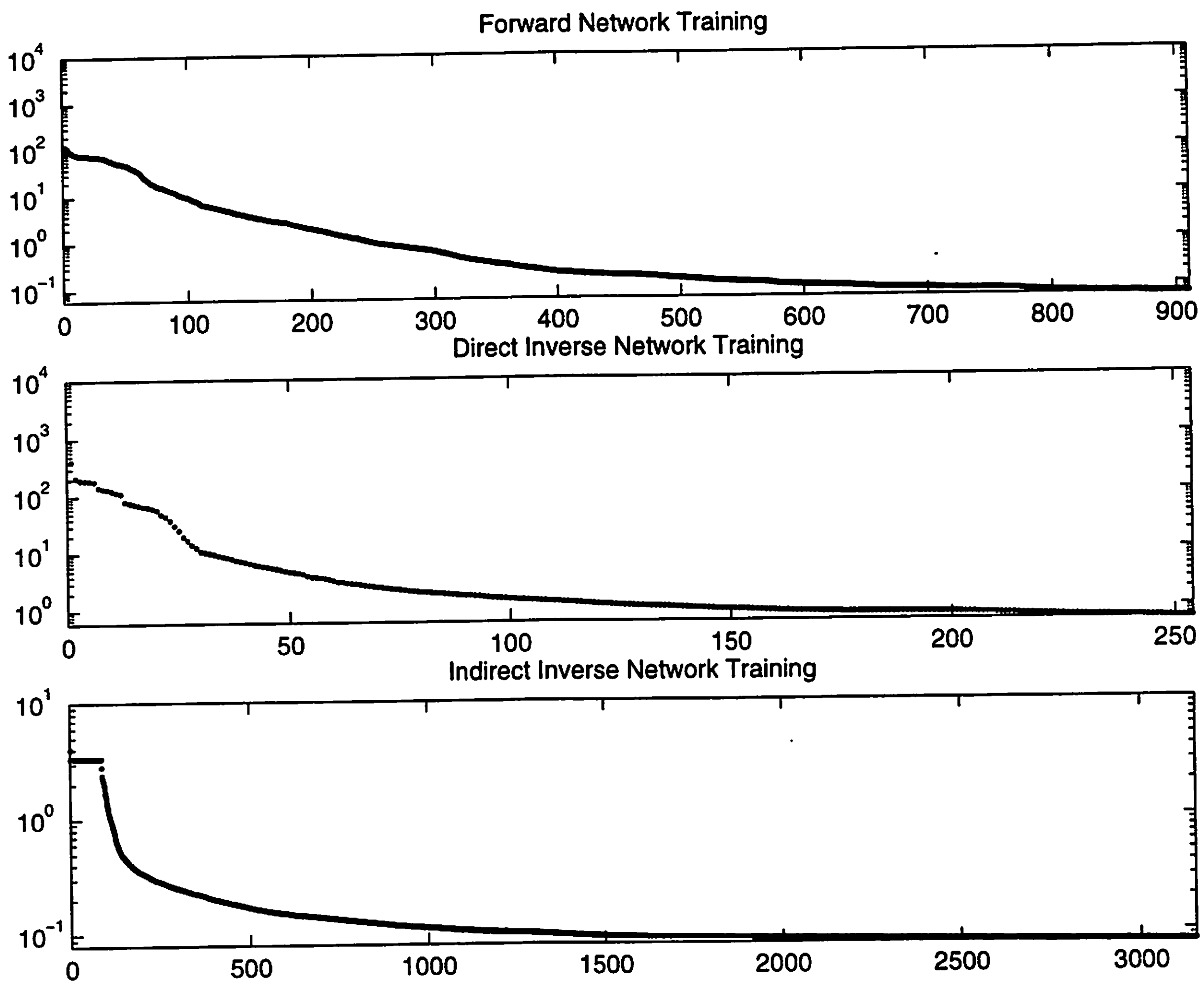


Figure C.1: Log (base 10) plots of the errors $\sum_p \sum_m (o_{p,m} - t_{p,m})^2 = 2\sum_p E_p$ of the DOF 0, PHM mode networks during training. Top: Forward network. The algorithm was run to a cutoff error of .08 after 733 iterations, then reinitialized and run to a cutoff of .06. Center: Inverse network, direct training phase. The algorithm was run to a cutoff error of .08 after 176 iterations, then reinitialized and run to a cutoff of .06. Bottom: Inverse network, indirect training phase. Error $2\sum_p E_p = \sum_p ||\hat{\mathbf{y}}_\mathbf{p} - \mathbf{y}_{\mathbf{p},\mathbf{d}}||$. The algorithm was run to a cutoff error of .08.

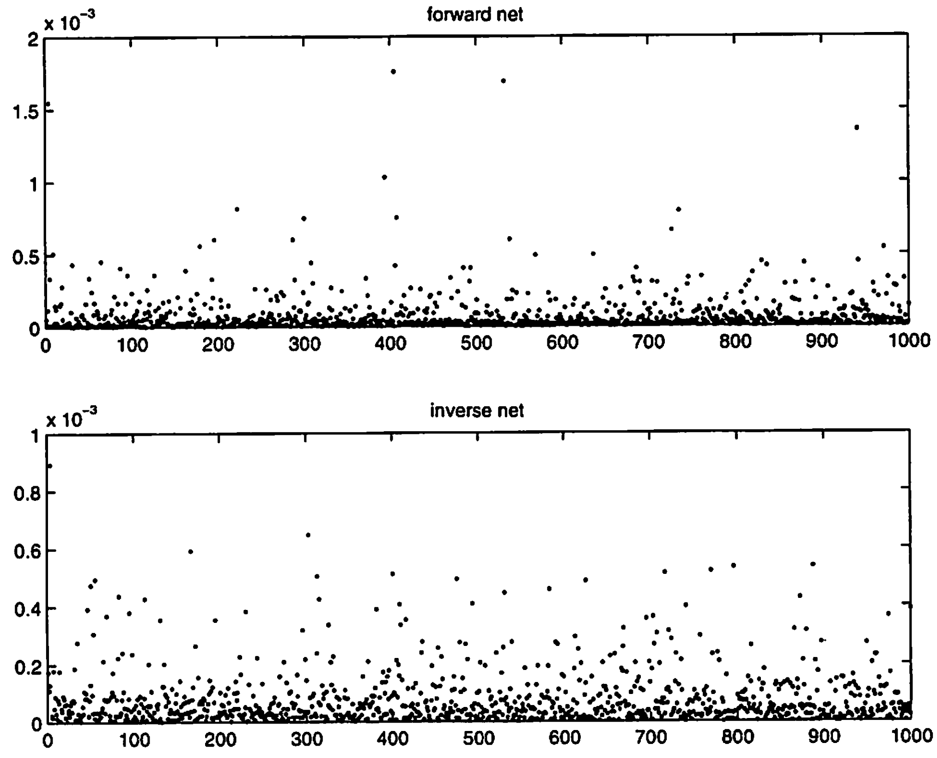


Figure C.2: Errors for the DOF 2, PWM mode networks after training. The top plot shows the error of the forward net on all 1000 training data samples, and the bottom plot shows the errors $||\hat{\mathbf{y}}_{\mathbf{p}} - \mathbf{y}_{\mathbf{p,d}}||^2$ for the inverse net and the forward net combined (indirect training method). The errors are calculated as $\sum_m (o_{p,m} - t_{p,m})^2 = 2E_p$, measured on scaled data.

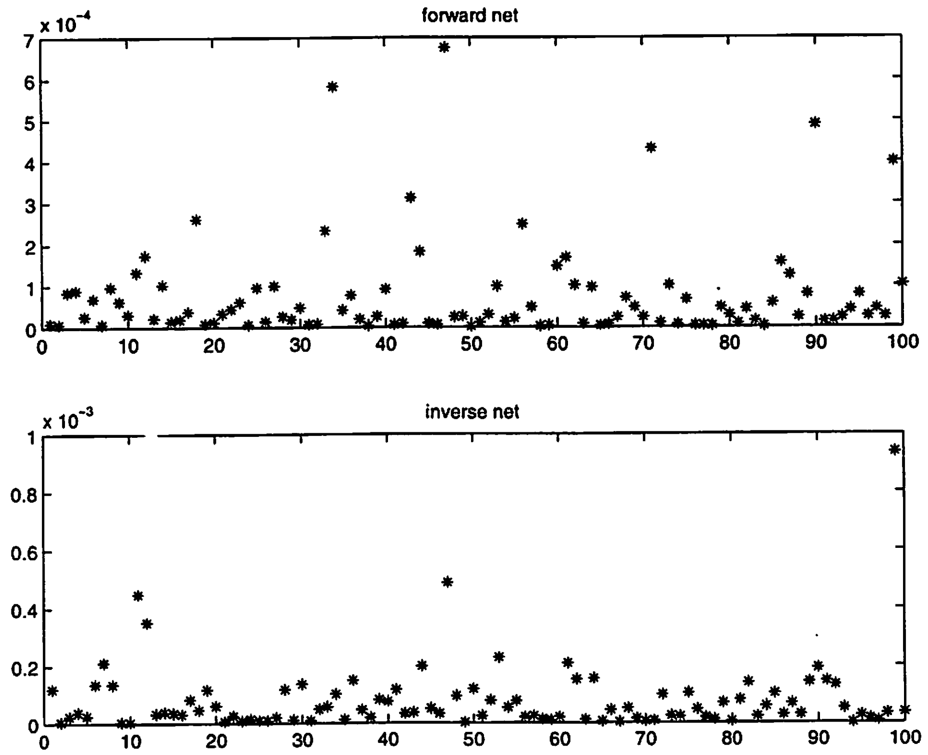


Figure C.3: Errors for the DOF 2, PWM mode networks when tested on a 100-point test data set. The top plot shows the forward net error, and the bottom plot shows the error $||\hat{\mathbf{y}}_\mathbf{p} - \mathbf{y}_{\mathbf{p},\mathbf{d}}||^2$ of the inverse net and the forward net combined. The errors are calculated as $\sum_m (o_{p,m} - t_{p,m})^2 = 2E_p$, measured on scaled data.

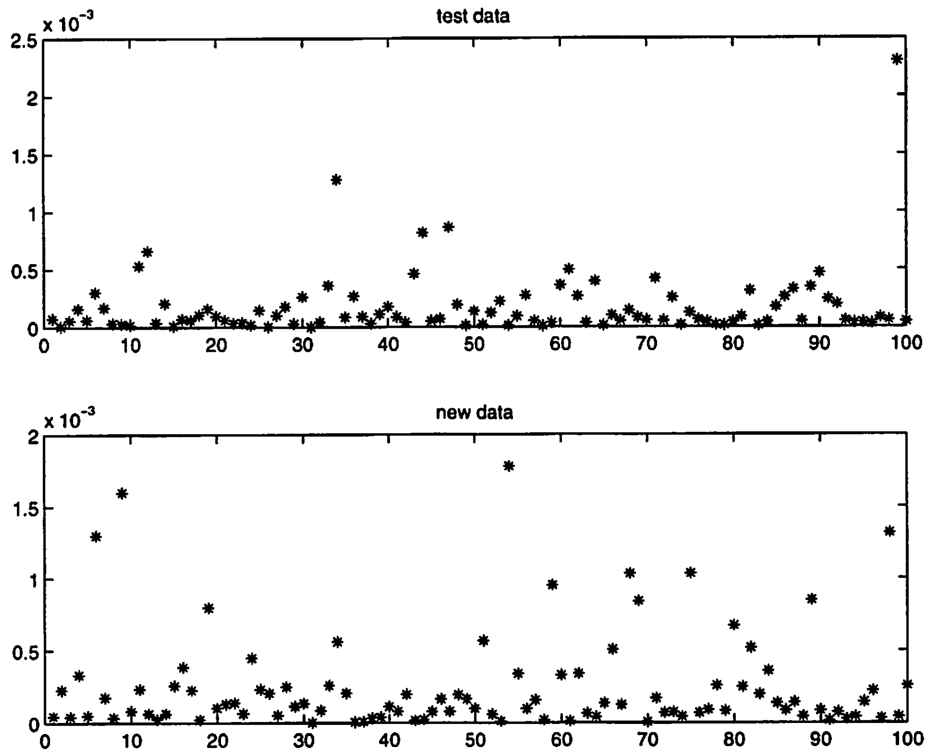
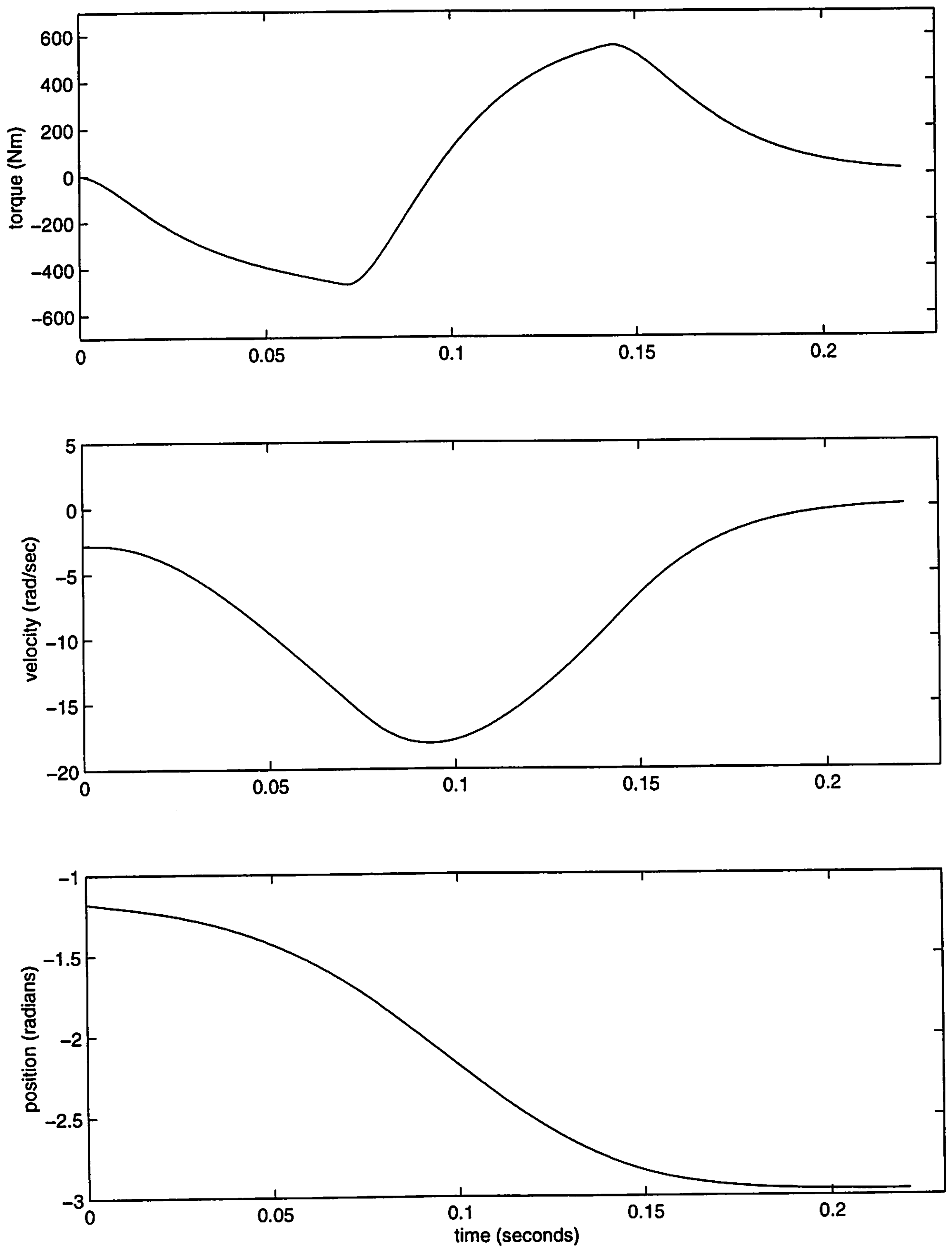
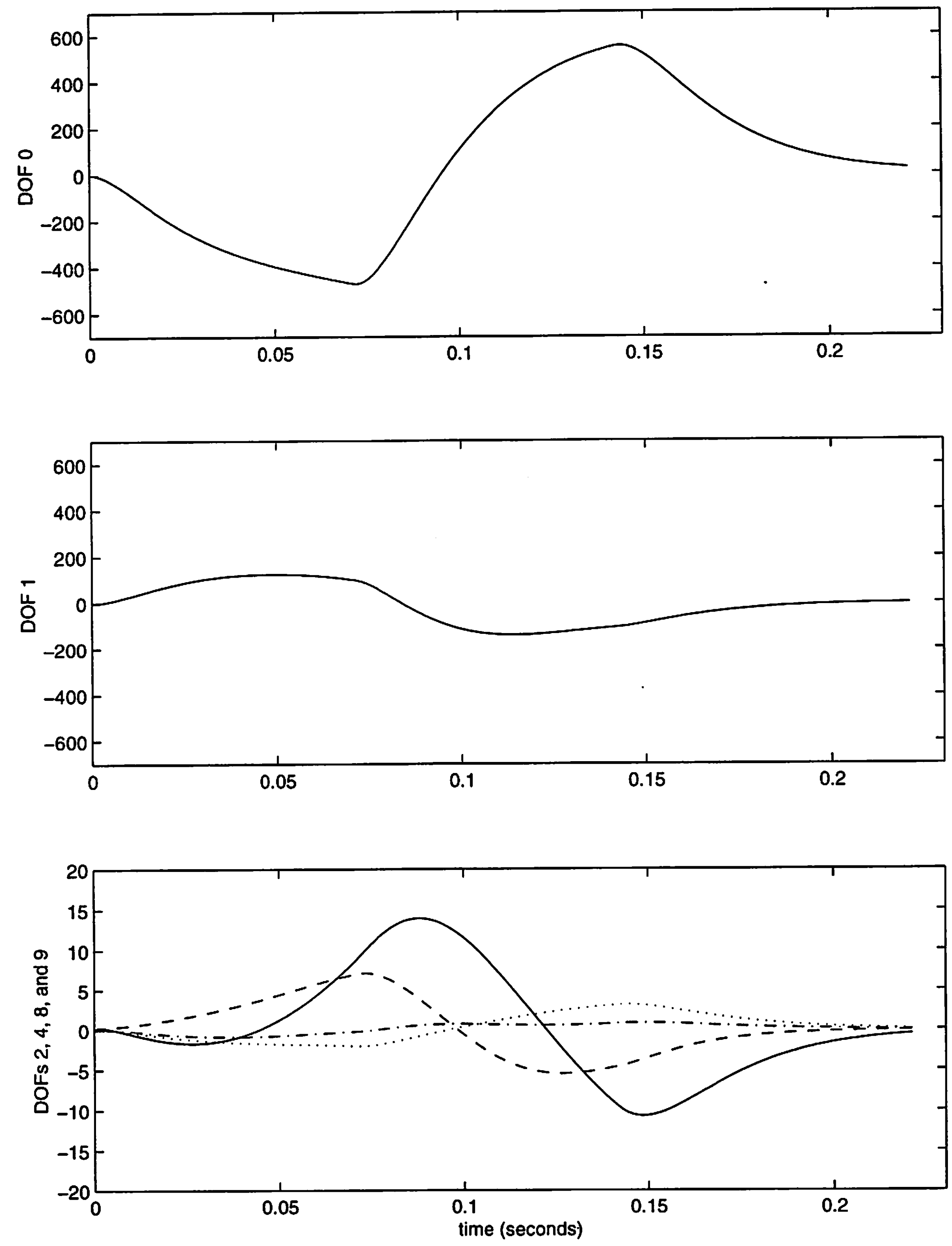


Figure C.4: Errors $||\hat{\mathbf{y}}_\mathbf{p} - \mathbf{y}_{\mathbf{p},\mathbf{d}}||^2$ for the DOF 2, PWM mode networks when controls generated by the inverse networks are simulated. The top plot shows the errors for the 100-point test data set, and the bottom plot shows the errors for 100 newly generated points which were tested for feasibility by being passed through the forward network (see text). The errors are calculated as $\sum_m (o_{p,m} - t_{p,m})^2 = 2E_p$, measured on scaled data.

A


600
400
200
0
-200
-400
-600
torque (Nm)
0
0.05
0.1
0.15
0.2
5
0
-5
-10
-15
-20
velocity (rad/sec)
0
0.05
0.1
0.15
0.2
-1
-1.5
-2
-2.5
-3
position (radians)
0
0.05
0.1
0.15
0.2
time (seconds)

B


600
400
200
0
−200
−400
−600
DOF 0
0
0.05
0.1
0.15
0.2
DOF 1
20
15
10
5
−5
−10
−15
−20
DOFs 2, 4, 8, and 9
time (seconds)

C

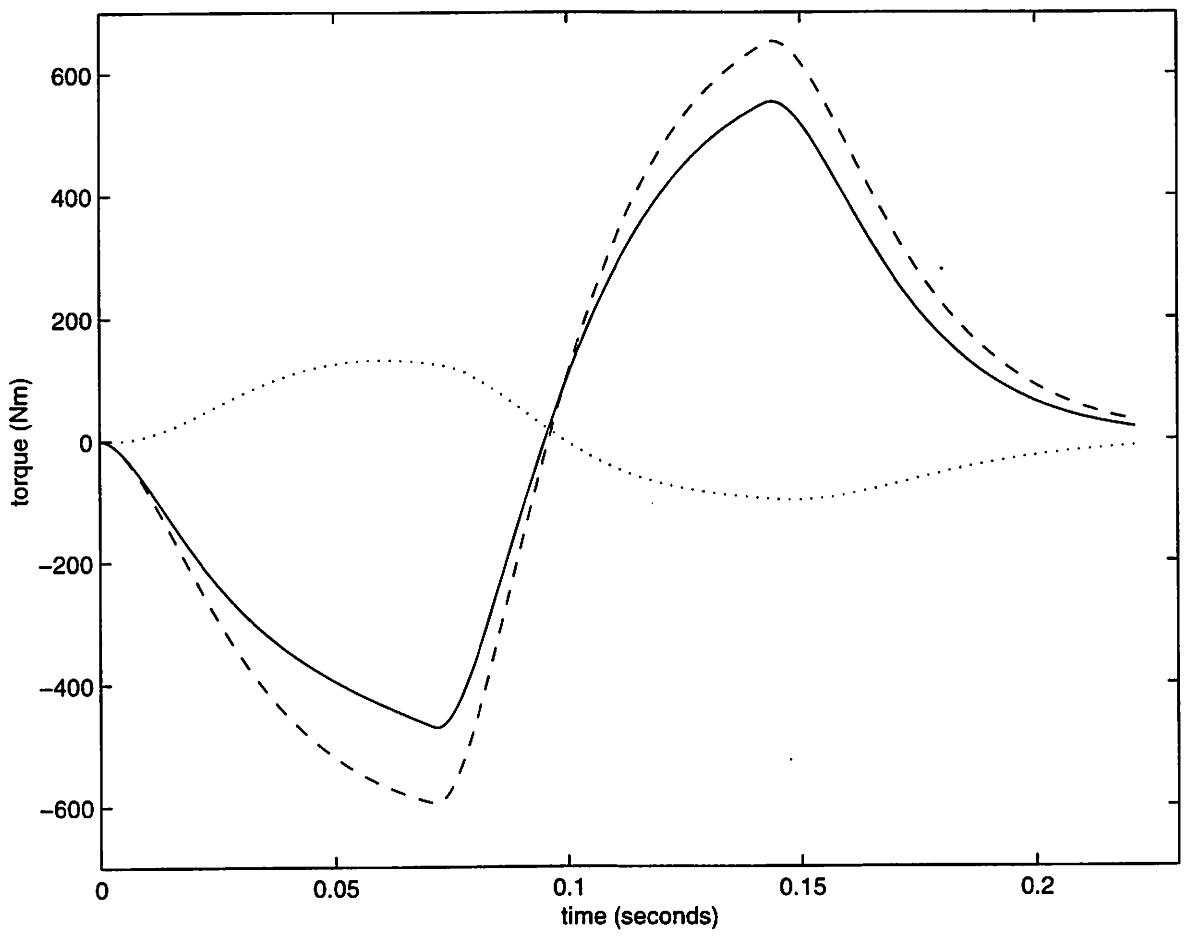
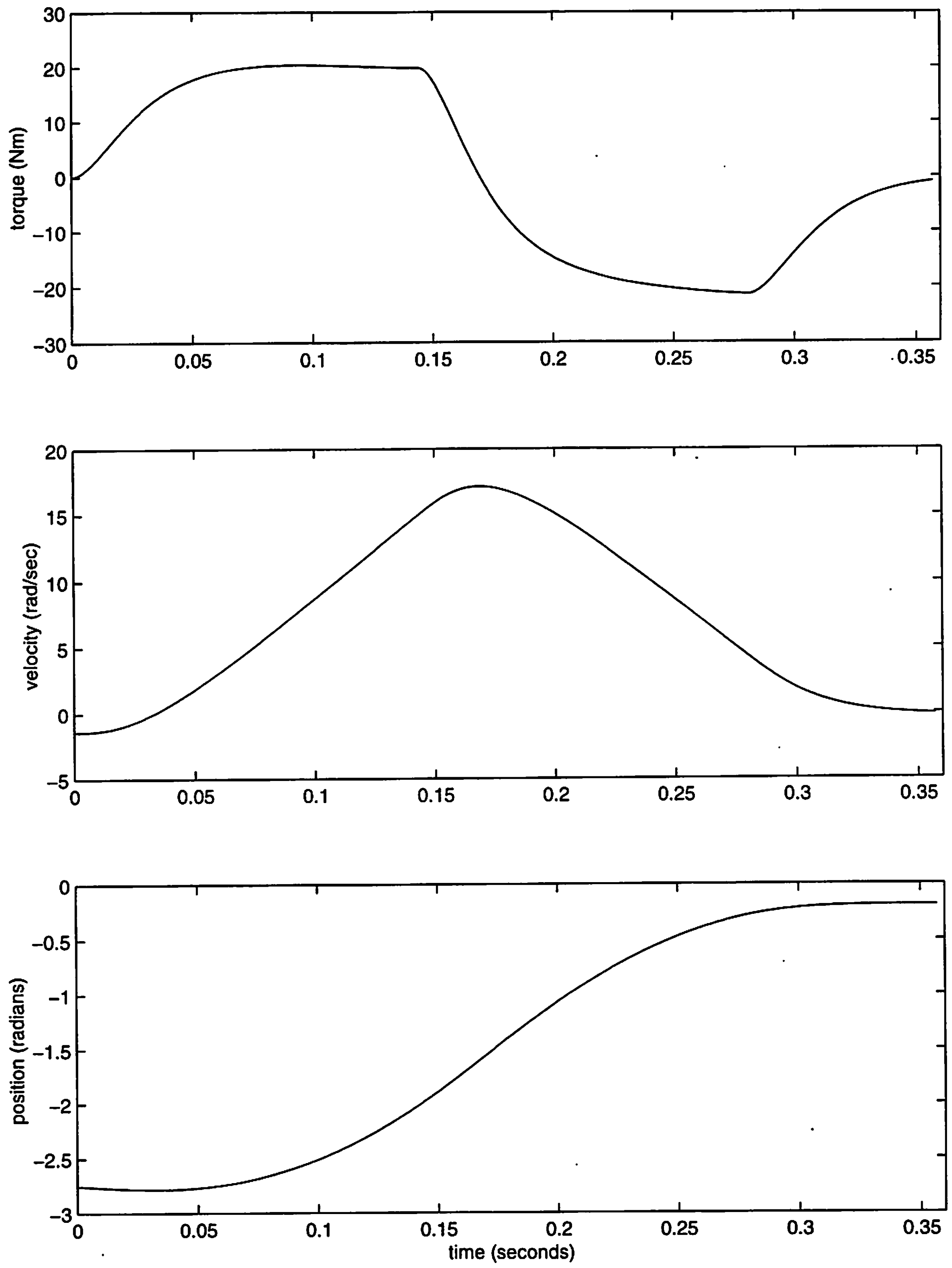


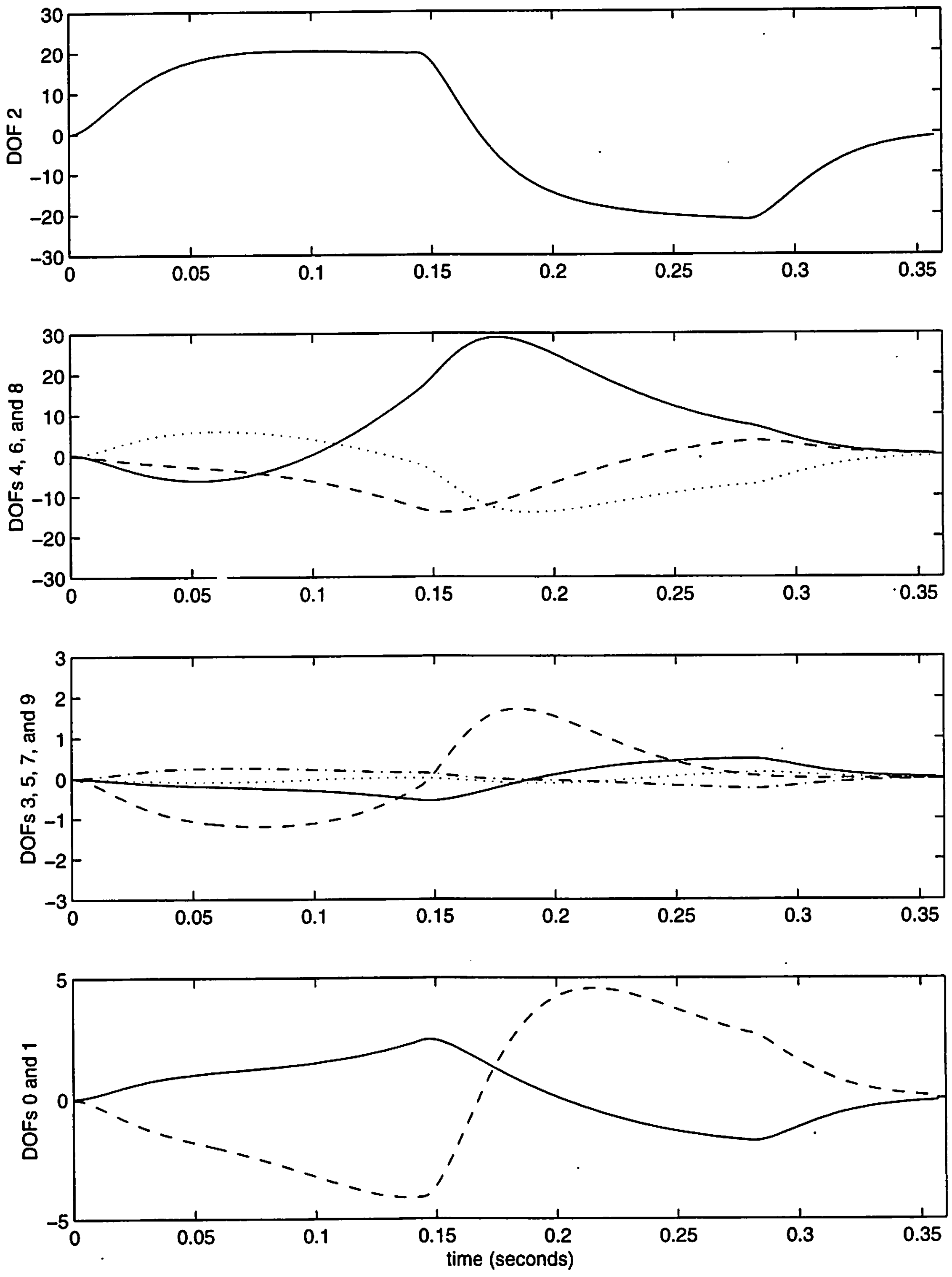
Figure C.5: A sample movement produced by the trained single-DOF controller for DOF 0 (right hip), PHM mode. The desired movement outcomes were $\Delta\theta_d = -1.885$ (scaled value of .2), $\Delta\dot{\theta}_d = 2.790$ (scaled value of .7), and $T_d = .220$ (scaled value of .2) seconds. v_0 was -2.790 (scaled value of .1). The scaled control values selected by the control network were phl=.174, ph2=.781, pw=.354. The actual movement outcomes were $\Delta\theta = -1.784$, $\Delta\dot{\theta} = 2.923$, and $T = .221$. The initial angle for DOF 0 was -1.179. The angles for the other degrees of freedom were chosen randomly, as discussed in the text. A. Torque, velocity, and position of DOF 0. B. Torques (in Newton-meters) for various degrees of freedom during the movement. All degrees of freedom other than 0 were held at a fixed angle. C. Components of the DOF 0 torque that were contributed by the feed-forward filtered pulses and the PD feedback.

A



torque (Nm)
30
20
10
0
−10
−20
−30
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
velocity (rad/sec)
20
15
10
5
0
−5
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
position (radians)
0
−0.5
−1
−1.5
−2
−2.5
−3
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
time (seconds)

B


DOF 2
30
20
10
0
−10
−20
−30
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
DOFs 4, 6, and 8
DOFs 3, 5, 7, and 9
3
2
1
−1
−2
−3
DOFs 0 and 1
5
−5
time (seconds)

C

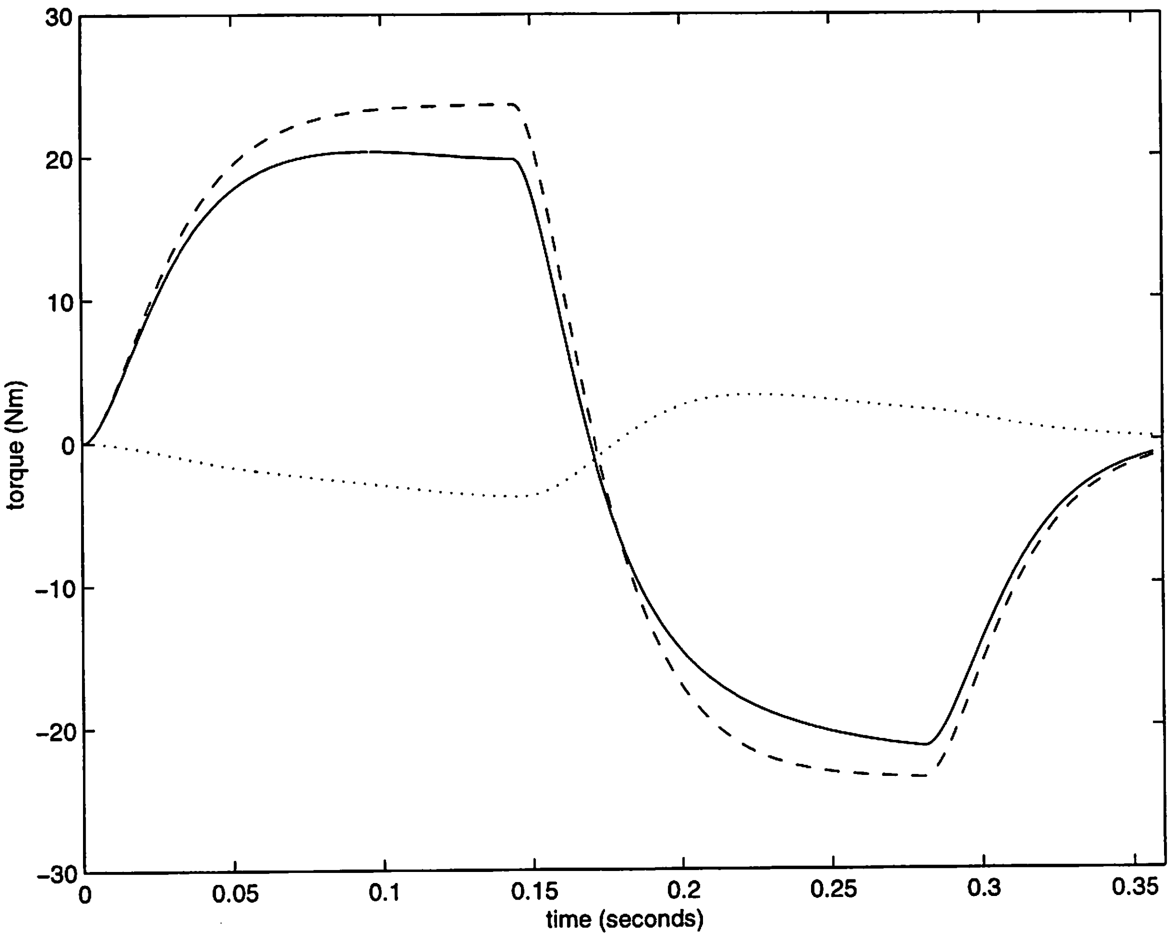
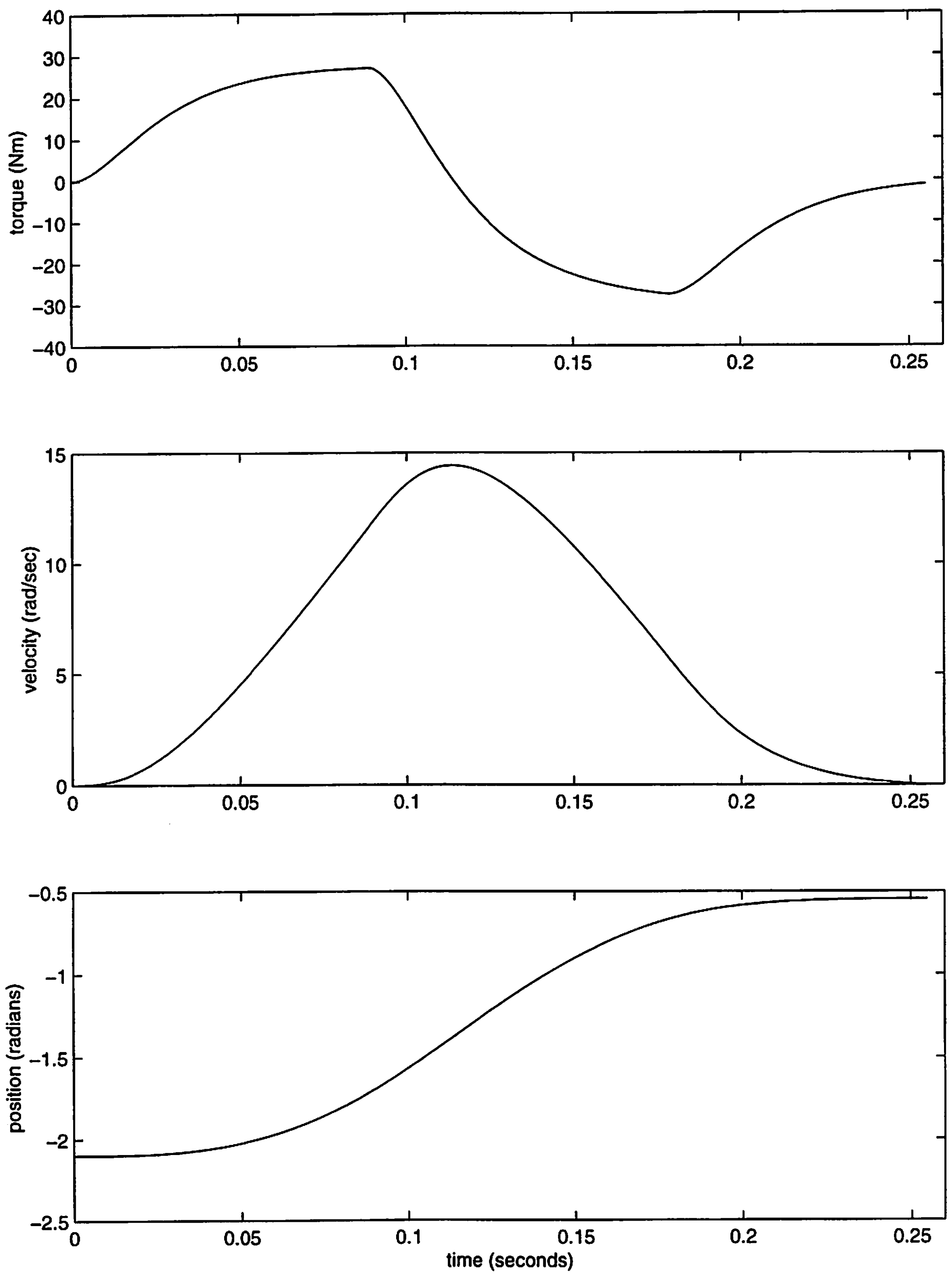
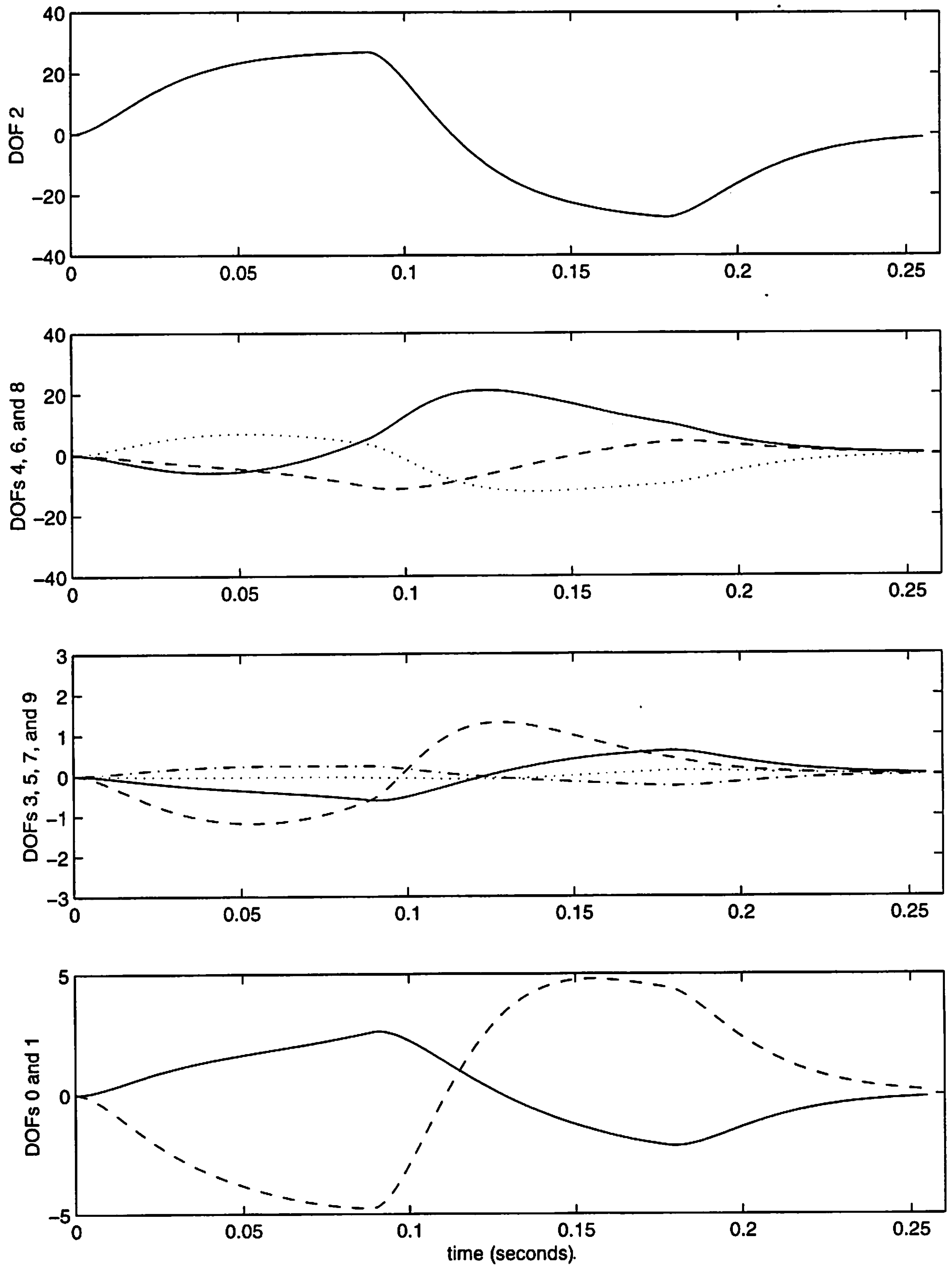


Figure C.6: A sample movement produced by the trained single-DOF controller for DOF 2 (the first Euler angle in the right shoulder), PWM mode. The desired movement outcomes were $\Delta\theta_d = 2.513$ (scaled value of .9), $\Delta\dot{\theta}_d = 1.395$ (scaled value of .6), and $T_d = .360$ (scaled value of .6) seconds. v_0 was -1.395 (scaled value of .3). The scaled control values selected by the control network were pwl=.745, pw2=.377, ph=.431. The actual movement outcomes were $\Delta\theta = 2.560$, $\Delta\dot{\theta} = 1.275$, and $T = .357$. The initial angle for DOF 2 was -2.752. The angles for the other degrees of freedom were chosen randomly, as discussed in the text. A. Torque, velocity, and position of DOF 2. B. Torques (in Newton-meters) for various degrees of freedom during the movement. All degrees of freedom other than 2 were held at a fixed angle. C. Components of the DOF 2 torque that were contributed by the feed-forward filtered pulses and the PD feedback.

A


torque (Nm)
40
30
20
10
0
−10
−20
−30
−40
0
0.05
0.1
0.15
0.2
0.25
velocity (rad/sec)
15
10
5
0
position (radians)
−0.5
−1
−1.5
−2
−2.5
time (seconds)

B


DOF 2
DOFs 4, 6, and 8
DOFs 3, 5, 7, and 9
DOFs 0 and 1
time (seconds)
40
20
0
-20
-40
3
2
1
-1
-2
-3
5
-5
0.05
0.1
0.15
0.2
0.25

C

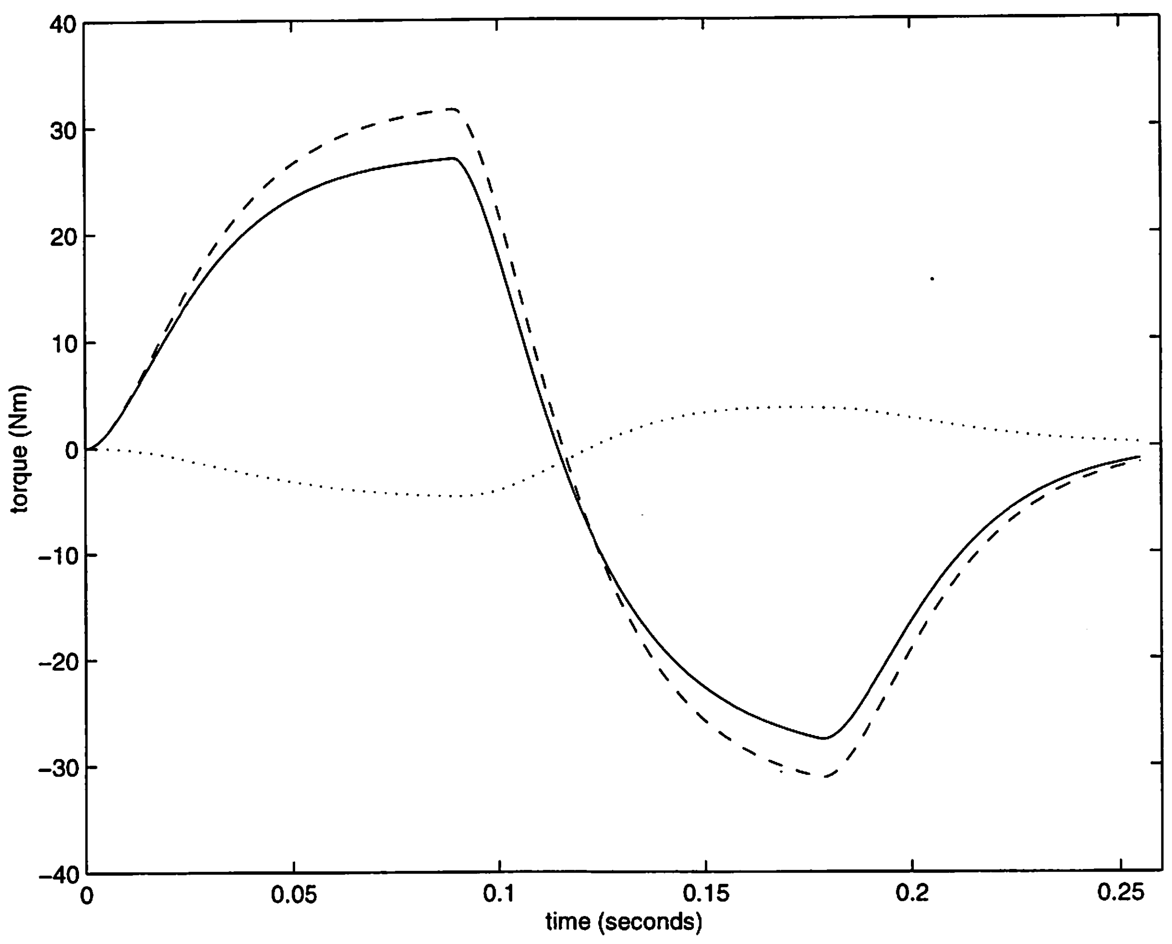


Figure C.7: A sample movement produced by the trained single-DOF controller for DOF 2 (the first Euler angle in the right shoulder), PHM mode. The desired movement outcomes were $\Delta\theta_d = 1.571$ (scaled value of .75), $\Delta\dot{\theta}_d = 0$ (scaled value of .5), and $T_d = .255$ (scaled value of .3) seconds. v_0 was zero (scaled value of .5). The scaled control values selected by the control network were phl=.703, ph2=.413, pw=.649. The actual movement outcomes were $\Delta\theta = 1.552$, $\Delta\dot{\theta} = -.00990$, and $T = .255$. The initial angle for DOF 2 was -2.098. The angles for the other degrees of freedom were chosen randomly, as discussed in the text. A. Torque, velocity, and position of DOF 2. B. Torques (in Newton-meters) for various degrees of freedom during the movement. All degrees of freedom other than 2 were held at a fixed angle. C. Components of the DOF 2 torque that were contributed by the feed-forward filtered pulses and the PD feedback.

Bibliography

[1] B. Abernethy and W. A. Sparrow. The rise and fall of dominant paradigms in motor behaviour research. In J. J. Summers, editor, *Approaches to the Study of Motor Control and Learning*. North-Holland, New York, 1992.

[2] M. Agarwal. A systematic classification of neural-network-based control. *IEEE Control Systems Magazine*, 17(2):75–93, April 1997.

[3] D. J. Aidley. *The Physiology of Excitable Cells.* Cambridge University Press, New York, 1989.

[4] M. A. Arbib. *The Metaphorical Brain 2: Neural Networks and Beyond.* Wiley, New York, 1989.

[5] C. G. Atkeson and J. M Hollerbach. Kinematic features of unrestrained vertical arm movements. *Journal of Neuroscience*, 5(9):2318–2330, 1985.

[6] J. Baillieul. Geometric methods for nonlinear optimal control problems. *Journal of Optimization Theory and Applications*, 25(4):519–548, 1978.

[7] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[8] B. W. Bequette. Nonlinear control of chemical processes: A review. *Industrial and Engineering Chemistry Research*, 30:1391–1413, 1991.

[9] N. E. Berthier, S. P. Singh, A. G. Barto, and J. C. Houk. Distributed representation of limb motor programs in arrays of adjustable pattern generators. *Journal of Cognitive Neuroscience*, 5(1):56–78, 1993.

[10] D. P. Bertsekas and J. N. Tsitsiklis. Neuro-dynamic programming: an overview. In *Proceedings of the 34th IEEE Conference on Decision and Control*, pages 560–564, 1995.

[11] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

[12] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Inc., New York, 1995.

[13] C. Bregler and J. Malik. Tracking people with twists and exponential maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1998.

[14] R. Brockett. On the computer control of movement. In *Proceedings of the IEEE Conference on Robotics and Automation*, 1988.

[15] R. Brockett. Formal languages for motion description and map making. In *Robotics.* American Mathematical Society, Providence, 1990.

[16] R. Brockett. Hybrid models for motion control systems. In Trentelman and Willems, editors, *Essays on Control: Perspectives in the Theory and its Applications.* Birkhauser, 1993.

[17] R. W. Brockett. Systems theory on group manifolds and coset spaces. *SIAM Journal of Control*, 10(2):265–284, 1972.

[18] R. A. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. *Neural Computation*, 1:253–262, 1989.

[19] R. A. Brooks. The role of learning in autonomous robots. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 5–10, 1991.

[20] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1996.

[21] V. Brooks. *The Neural Basis of Motor Control.* Oxford University Press, New York, 1986.

[22] L. Bushnell, D. Tilbury, and S. Sastry. Extended goursat normal forms with applications to nonholonomic motion planning. In *Proceedings of the 32nd IEEE Conference on Decision and Control*, volume 4, pages 3447–3452, 1993.

[23] J. J. Collins and I. N. Stewart. Coupled nonlinear oscillators and the symmetries of animal gaits. *Journal of Nonlinear Science*, 3:349–392, 1993.

[24] L. S. Crawford and S. S. Sastry. Biological motor approaches for a planar diver. In *Proceedings of the 34th IEEE Conference on Decision and Control*, pages 3881–3886, December 1995.

[25] C. T. Farley and D. P. Ferris. Biomechanics of walking and running: center of mass movements to muscle action. *Exercise and Sports Sciences Reviews*, 1998. In press.

[26] P. M. Fitts. Factors in complex skill training. In R. Glaser, editor, *Training Research and Education*. University of Pittsburgh Press, Pittsburgh, 1962.

[27] C. Frohlich. Do springboard divers violate angular momentum conservation? *American Journal of Physics*, 47(7):583–592, July 1979.

[28] P. Di Giamberardino, S. Monaco, and D. Normand-Cyrot. Digital control through finite feedback discretizability. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, pages 3141–3146, 1996.

[29] S. Giszter. Spinal movement primitives and motor programs–a necessary concept for motor control. *Behavioral and Brain Sciences*, 15(4):744–745, 1992.

[30] D. J. Glencross, H. T. A. Whiting, and B. Abernethy. Motor control, motor learning and the acquisition of skill: historical trends and future directions. *International Journal of Sport Psychology*, 25:32–52, 1994.

[31] J-M. Godhavn, A. Balluchi, L. S. Crawford, and S. S. Sastry. Control of nonholonomic systems with drift terms. *Automatica*, to appear.

[32] D. Gorinevsky, A. Kapitanovsky, and A. Goldenberg. Radial basis function network architecture for nonholonomic motion planning and control of free-flying manipulators. *IEEE Transactions on Robotics and Automation*, 12(3), June 1996.

[33] G. L. Gottlieb. A computational model of the simplest motor program. *Journal of Motor Behavior*, 25(3):153–161, 1993.

[34] G. L. Gottlieb, D. M. Corcos, and G. C. Agarwal. Strategies for the control of voluntary movements with one mechanical degree of freedom. *Behavioral and Brain Sciences*, 12:189–210, 1989.

[35] S. Grillner. Locomotion in vertebrates: central mechanisms and reflex interaction. *Physiological Reviews*, 55(2):247–304, April 1975.

[36] S. Grillner and P. Wallén. Central pattern generators for locomotion, with special reference to vertebrates. *Annual Review of Neuroscience*, 8:233–261, 1985.

[37] B. Hannaford and L. Stark. Roles of the elements of the triphasic control signal. *Experimental Neurology*, 90:619–634, 1985.

[38] S. S. Haykin. *Neural Networks: A Comprehensive Foundation.* MacMillan, New York, 1994.

[39] J. Hertz, Anders Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley Publishing Company, Redwood City, California, 1991.

[40] A. V. Hill. The heat of shortening and the dynamic constants of muscle. *Proceedings of the Royal Society of London, Series B, Biological Sciences*, 126:136–195, 1938.

[41] J. K. Hodgins. Three-dimensional human running. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3271–3276, 1996.

[42] J. K. Hodgins and N. S. Pollard. Adapting simulated behaviors for new characters. In *Computer Graphics Proceedings, SIGGRAPH 97*, pages 153–162, 1997.

[43] D. S. Hoffman and P. L. Strick. Step-tracking movements of the wrist in humans. I. Kinematic analysis. *Journal of Neuroscience*, 6(11):3309–3318, 1986.

[44] D. S. Hoffman and P. L. Strick. Step-tracking movements of the wrist in humans. II. EMG analysis. *Journal of Neuroscience*, 10(1):142–152, 1990.

[45] M. G. Hollars, D. E. Rosenthal, and M. A. Sherman. *SD/FAST User's Manual, Version B.2.* Mountain View, CA, 1994.

[46] J. M. Hollerbach and C. G. Atkeson. Deducing planning variables from experimental arm trajectories: pitfalls and possibilities. *Biological Cybernetics*, 56:279–292, 1987.

[47] J. M. Hollerbach and C. G. Atkeson. Inferring limb coordination strategies from trajectory kinematics. *Journal of Neuroscience Methods*, 21:181–194, 1987.

[48] D. Hong, D. M. Corcos, and G. L. Gottlieb. Task dependent patterns of muscle activation at the shoulder and elbow for unconstrained arm movements. *Journal of Neurophysiology*, 71(3):1261–1265, March 1994.

[49] J. C. Houk. Regulation of stiffness by skeletomotor reflexes. *Annual Review of Physiology*, 41:99–114, 1979.

[50] A. F. Huxley. Muscle structure and theories of contraction. *Progress in Biophysics*, 7:257–318, 1957.

[51] M. Ito. Neural systems controlling movement. *Trends in Neural Science*, pages 515–518, October 1986.

[52] M. Jeannerod. *The Neural and Behavioral Organization of Goal-Directed Movements.* Clarendon Press, Oxford, 1988.

[53] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[54] M. Kawato. Computational schemes and neural network models for formation and control of multijoint arm trajectory. In W. T. Miller III, R. S. Sutton, and P. J. Werbos, editors, *Neural Networks for Control*, pages 197–228. MIT Press, Cambridge, MA, 1990.

[55] M. Kawato. Feedback-error-learning neural network for supervised motor learning. In R. Eckmiller, editor, *Advanced Neural Computers*, pages 365–372. North-Holland, Amsterdam, 1990.

[56] M. Kawski. Nilpotent Lie algebras of vectorfields. *Journal für die reine und angewandte Mathematik*, 388:1–17, 1988.

[57] M. Kawski. Combinatorics of realizations of nilpotent control systems. In M. Fliess, editor, *Nonlinear Control Systems Design 1992: Selected papers from the 2nd IFAC symposium*, pages 251–256. Pergamon Press, Oxford, 1993.

[58] I. Kolmanovsky and N. H. McClamroch. Developments in nonholonomic control problems. *IEEE Control Systems Magazine*, 15(6):20–36, December 1995.

[59] W. S. Levine and G. E. Loeb. The neural control of limb movement. *IEEE Control Systems Magazine*, 12(6):38–47, December 1992.

[60] D. G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, Reading, MA, 1984.

[61] P. Maes and R. A. Brooks. Learning to coordinate behaviors. In *AAAI-90 Proceedings*, volume 2, pages 796–802, 1990.

[62] V. Manikonda, P. S. Krishnaprasad, and J. Hendler. Languages, behaviors, and motion control. *International Journal of Robotics Research*, 1998. to appear.

[63] D. Q. Mayne. Nonlinear model predictive control: An assessment. In *Chemical Process Control-V, Proceedings of the Fifth International Conference on Chemical Process Control*, pages 217–231. American Institute of Chemical Engineers, New York, 1997.

[64] T. A. McMahon. *Muscles, Reflexes, and Locomotion.* Princeton University Press, Princeton, 1984.

[65] H.-M. Meinck, R. Benecke, W. Meyer, J. Hohne, and B. Conrad. Human ballistic finger flexion: uncoupling the three-burst pattern. *Experimental Brain Research*, 55:127–133, 1984.

[66] J. P. Miller. Pyloric mechanisms. In A. I. Selverston and M. Moulins, editors, *The Crustacean Stomatogastric System*, pages 109–145. Springer-Verlag, Berlin, 1987.

[67] J. F. Montgomery and G. A. Bekey. Learning helicopter control through "teaching by showing". In *Proceedings of the 37th IEEE Conference on Decision and Control*, 1998. To appear.

[68] R. Montgomery. Isoholonomic problems and some applications. *Communications in Mathematical Physics*, 128(3):565–592, 1990.

[69] K. L. Moore. *Iterative Learning Control for Deterministic Systems.* Springer-Verlag, New York, 1993.

[70] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation.* CRC Press, Boca Raton, Florida, 1994.

[71] R. M. Murray and S. S. Sastry. Nonholonomic motion planning: steering using sinusoids. *IEEE Transactions on Automatic Control*, 38(5):700–716, May 1993.

[72] K. M. Newell. Motor skill acquisition. *Annual Review of Psychology*, 42:213–237, 1991.

[73] K. M. Newell, P. N. Kugler, R. E. A. Van Emmerik, and P. V. McDonald. Search strategies and the acquisition of coordination. In S. A. Wallace, editor, *Perspectives on the Coordination of Movement*, pages 85–122. North-Holland, New York, 1989.

[74] D. Nguyen and B. Widrow. The truck backer-upper: An example of self-learning in neural networks. In W. T. Miller III, R. S. Sutton, and P. J. Werbos, editors, *Neural Networks for Control*, pages 287–299. MIT Press, Cambridge, MA, 1990.

[75] D. H. Owens, N. Amann, and E. Rogers. Iterative learning control – an overview of recent algorithms. *Applied Math and Computer Science*, 5(3):425–438, 1995.

[76] George J. Pappas, John Lygeros, Dawn Tilbury, and Shankar Sastry. Exterior differential systems in control and robotics. In J. Baillieul, S. Sastry, and H. Sussmann, editors, *Essays on Mathematical Robotics*, volume 104 of *IMA Volumes in Mathematics and its Applications.* Springer Verlag, New York, 1998. To appear.

[77] A. Pedotti, V. V. Krishnan, and L. Stark. Optimization of muscle-force sequencing in human locomotion. *Mathematical Biosciences*, 38:57–76, 1978.

[78] A. C. Pil and H. H. Asada. Integrated structure/control design of mechatronic systems using a recursive experimental optimization method. *IEEE/ASME Transactions on Mechatronics*, 1(3):191–203, September 1996.

[79] M. H. Raibert. *Legged Robots That Balance.* MIT Press, Cambridge, MA, 1986.

[80] M. H. Raibert and J. K. Hodgins. Animation of dynamic legged locomotion. *Computer Graphics*, 25(4):349–358, July 1991.

[81] C. Ramos, L. Stark, and B. Hannaford. Time optimality, proprioception, and the triphasic EMG pattern. *Behavioral and Brain Sciences*, 12(2):231–232, 1989.

[82] C. F. Ramos, S. S. Hacisalihzade, and L. W. Stark. Behaviour space of a stretch reflex model and its implications for the neural control of voluntary movement. *Medical and Biological Engineering and Computing*, 28:15–23, 1990.

[83] J. B. Rawlings, E. S. Meadows, and K. R. Muske. Nonlinear model predictive control: A tutorial and survey. In D. Bonvin, editor, *Advanced Control of Chemical Processes (ADCHEM '94): IFAC Symposium*, pages 185–197. Pergamon, Tarrytown, NY, 1994.

[84] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Englewood Cliffs, NJ, 1995.

[85] A. Sarti, G. Walsh, and S. Sastry. Steering left-invariant control systems on matrix Lie groups. In *Proceedings of the 32nd IEEE Conference on Decision and Control*, volume 4, pages 3117–3121, 1993.

[86] S. Sastry and M. Bodson. *Adaptive Control: Stability, Convergence, and Robustness.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[87] S. S. Sastry and R. Montgomery. The structure of optimal controls for a steering problem. In M. Fliess, editor, *Nonlinear Control Systems Design 1992*, pages 135–140. Pergamon Press, Oxford, 1993.

[88] S. Schaal. Learning from demonstration. In M. C. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 1040–1046, Cambridge, MA, 1997. MIT Press.

[89] R. A. Schmidt. A schema theory of discrete motor skill learning. *Psychological Review*, 82(4):225–260, 1975.

[90] R. A. Schmidt. *Motor Control and Learning.* Human Kinetics Publishers, Champaign, IL, 1982.

[91] Schöner and Kelso. A synergetic theory of environmentally-specified and learned patterns of movement coordination: I. relative phase dynamics. *Biological Cybernetics*, 58:71–80, 1988.

[92] R. Shadmehr and F. A. Mussa-Ivaldi. Adaptive representation of dynamics during learning of a motor task. *Journal of Neuroscience*, 14(5):3208–3224, May 1994.

[93] J. F. Soechting. Elements of coordinated arm movements in three-dimensional space. In S. A. Wallace, editor, *Perspectives on the Coordination of Movement*, pages 47–83. North-Holland, New York, 1989.

[94] E. D. Sontag. Feedback stabilization using two-hidden-layer nets. *IEEE Transactions on Neural Networks*, 3(6):981–990, November 1992.

[95] G. Strang. Wavelets and dilation equations: A brief introduction. *SIAM Review*, 31:613–627, 1989.

[96] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[97] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

[98] G. Tesauro. TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

[99] E. Thelen. The development of leg coordination. In S. A. Wallace, editor, *Perspectives on the Coordination of Movement*, pages 259–281. North-Holland, New York, 1989.

[100] D. Tilbury, R. M. Murray, and S. S. Sastry. Trajectory generation for the N-trailer problem using Goursat normal form. *IEEE Transactions on Automatic Control*, 40(5):802–819, May 1995.

[101] B. Vereijken, R. E. A. Van Emmerik, H. T. A. Whiting, and K. M. Newell. Free(z)ing degrees of freedom in skill acquisition. *Journal of Motor Behavior*, 24(1):133–142, March 1992.

[102] B. Vereijken, H. T. A. Whiting, and Beek. A dynamical systems approach to skill acquisition. *Quarterly Journal of Experimental Psychology Section A – Human Experimental Psychology*, 45(2):323–344, August 1992.

[103] W. J. Wadman, J. J. Denier van der Gon, R. H. Geuze, and C. R. Mol. Control of fast goal-directed arm movements. *Journal of Human Movement Studies*, 5:3–17, 1979.

[104] G. Walsh, A. Sarti, and S. Sastry. Algorithms for steering on the group of rotations. In *Proceedings of the 1993 American Control Conference*, volume 2, pages 1312–1316, 1993.

[105] G. C. Walsh and S. S. Sastry. On reorienting linked rigid bodies using internal motions. *IEEE Transactions on Robotics and Automation*, 11(1):139–146, February 1995.

[106] P. J. Werbos. An overview of neural networks for control. In V. R. Vemuri, editor, *Artificial Neural Networks: Concepts and Control Applications.* IEEE Computer Society Press, Los Alamitos, CA, 1992.

[107] H. T. A. Whiting, editor. *Human Motor Actions: Bernstein Reassessed.* Elsevier Science Publishing Company, New York, 1984.

[108] D. A. Winter. Coordination of motor tasks in human gait. In S. A. Wallace, editor, *Perspectives on the Coordination of Movement*, pages 329–363. North-Holland, New York, 1989.

[109] D. A. Winter. Human movement: A system-level approach. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 472–477. MIT Press, Cambridge, MA, 1995.

[110] W. L. Wooten and J. K. Hodgins. Animation of human diving. *Computer Graphics Forum*, 15(1):3–13, March 1996.