# Chapter 7

# Autonomous Robot Motion Control

## 7.1 Introduction

It was seen in the previous chapter that FOX can successfully learn to control a variety of mechanical systems. These systems were simple, in the sense that their desired behavior was easily specified (e.g. keep the inverted pendulum upright, or keep the robot on the track). Thus FOX could control the system by optimizing a single error value. This is adaptive control, but it is not *intelligent* control.

This chapter explores the more difficult problem of learning to control an autonomous robot that has to survive in a complex environment, and whose desired behavior is only known in vague terms. Again, note that only the motion control aspect of this problem will be explored (control of high level behavior is a separate issue). Through trial and error a variety of design principles have been developed that are useful for designing such systems using FOX. These will be applied to controlling a hopping monoped robot and a walking biped robot. These robots are simulated, not real. The virtual environment that they inhabit will also be described. Several MPEG movies have been created that illustrate various aspects of robot motion and training. These movies are stored on the CDROM that accompanies this thesis. They will be referred to thus: movie `foo.mpg`.

## 7.2 Specifying behavior

An autonomous robot designer often only has a vague idea of the behavior that is required. For a wheeled robot the challenge is to intelligently control the high level aspects of behavior, for example specifying the direction and speed of the robot over time to achieve some task. Legged robots must use intelligent control at a much lower level, because the control problem is dynamically complex and it is not obvious what leg trajectories will be effective.

### 7.2.1 Hard-wired and generic controllers

Controllers for autonomous robots can usually be classified between two extremes: hard-wired and generic. A hard-wired controller is simply one that has been constructed to implement the desired behavior (and no other). It has no adjustable parameters (that is, nothing to learn), but it is pre-adapted to its environment. Examples include Beer's artificial insect [15] and Brooks' hexapod [20].

A generic controller is at the other extreme: it contains neural networks or some other structures which are capable of implementing arbitrary control rules. The desired behavior must be specified using some error or reward signal, as the controller starts out by knowing nothing. It has many parameters, all of which must be learned as the robot interacts with its environment. Examples include Millan's TESEO system [81] and MLP controllers such as [102].

A practical system needs to compromise between these two extremes. A fully hard wired approach can not adapt to unanticipated variations in its environment. A fully generic approach may take too long to train (or may have to be trained offline, or may be incapable of learning adequate control rules, see below). A partially hard-wired controller with a range of adjustable parameters should be designed. Thus best use of the designer's expert knowledge about the problem domain is made by spreading it between the hard-wired controller design and the choice of error/reward signals. This is also usually more time efficient than either extreme: parameters can be learned that would otherwise have to be adjusted by the designer through many design iterations in the hard wired approach.

The number of parameters to learn is a design decision that depends on the problem: more free parameters makes the controller more flexible, but can also increase the difficulty of learning.

### 7.2.2 Generic controllers using scalar error signals

The desired behavior is commonly defined as that which minimizes a simple scalar error value. This is the approach for which the FOX controller has been formulated, and the one that was used in the previous chapter. Three common approaches for making use of scalar errors are gradient descent, genetic algorithms and reinforcement learning. These techniques are used more for generic controllers. Consider, for example, how a biped robot could be made to walk where the error signal just tried to maximize the robot's forward speed. This error signal does not give any indication of *how* the task is to be achieved, for example it does not constrain the legs to move in a stepping pattern. In general, for more complicated systems a scalar error provides less information about what each component of the system should be doing.

It is common for the error gradient vector (with respect to the system control parameters) to be known, in which case a gradient descent technique can be used (such as conjugate gradient, see [100]). However gradient descent may be unable to discover complex control actions in a completely generic controller, as it relies on the existence of a continuous (or analytic) search space, but most interesting problems will be discontinuous. Also gradient descent can easily become trapped in local minima and fail to find any acceptable behavior, especially with complex systems [49]. Generic-controller gradient descent techniques are most useful in simple systems.

Genetic algorithms [36] are a class of techniques that are capable of discovering unique controller structures and patterns of movement using simple error signals [111, 112]. They are extremely flexible and they do not become trapped in local minima, but they do not easily lend themselves to on-line learning in real robots (as large populations of automatons are required). Also they can be extremely slow.

Reinforcement learning systems such as temporal difference [117] and *Q-learning* [69] can discover low error actions directly. These methods have a strong theoretical foundation. Typical implementations rely solely on scalar reinforcement signals from the controlled system to train a homogeneous neural network. Although in principle they can learn control strategies which minimize penalty, in practice their lack of behavioral constraints make them slow to converge and very computationally intensive to train.

### 7.2.3 Analytical motion

Many authors have described algorithms for generating limb trajectories that achieve some goal. For example, [75] describes pattern generating algorithms that produce obstacle avoiding limb trajectories, and [127] describes how recursive workspace multi-body dynamics techniques are used to control a planar biped. Such analytical methods require good models of the robot to be available (its exact geometry and mass distribution), and they can sometimes be very computationally intensive.

## 7.3 General design principles

Through much trial and error a number of principles have been found to be useful when designing autonomous robot controllers. First, a "toolbox" approach should be used—that is, don't insist on using just one paradigm (e.g. neural networks, reinforcement learning, or symbolic AI) because each approach has its advantages and disadvantages. Hybrid designs which use a variety of techniques can often be more robust.

The robot's behavioral repertoire can be divided amongst multiple interacting modules, possibly ordered in a hierarchy. The robot's behavior *emerges* from the interaction of these modules. This approach can allow the designer to be vague about the behavior required. The controller must learn low-penalty actions within the constraints of the behaviors defined by its internal structure.

A compromise is made between hard-wired and fully learned behavior. The designer's domain knowledge is used to design a controller that has the potential to adapt the robot well to its environment. Some parameters are left unspecified (maybe they are difficult to find beforehand, or perhaps they depend on the specifics of the robot's environment). These parameters are learned by FOX modules, which work independently (or together in cases where a single parameter has multiple constraints). The error signals for the FOX modules are selected independently, they are not derived from a global performance error. Thus the whole-body optimal behavior is not predefined, but nevertheless the behavior should improve as learning occurs. This approach will obviously work better if the parameters have relatively independent influences on the system behavior. This approach can not learn fundamentally new behavior, but it can improve existing behavior, which will be quite adequate in many situations.

Note that unlike MLPs, CMACs can not discover new internal representations (in other words they do not perform global generalization), and so the CMAC parameters must be chosen carefully beforehand.

Some more specific design techniques will be described later in this chapter.

## 7.4 The simulation and virtual environment

The hopping and walking robots were simulated and visualized using a suite of applications developed in C++ by the author. Full details are given in Appendix H, but briefly:

- `RoboDyn` is a C++ class library that simulates the dynamics (i.e. the motion) of articulated rigid bodies. The numerical engine is based on the work of Scott McMillan [76, 77, 78, 79]. RoboDyn provides very efficient simulation: it uses Featherstone's recursive articulated-body algorithm [34] which executes in $O(n)$ time where $n$ is the number of links in the system. Further details are given in Appendix I.

- `Dyson` is a language and compiler for specifying arbitrary dynamical systems. It was used to implement the robot controllers. It has the following components:

| Constant | Value |
|---|---|
| Gravitational constant | 9.81 $m/s^2$ |
| Ground planar spring constant | 0 N/m |
| Ground normal spring constant | 10000 N/m |
| Ground planar damper constant | 250 Ns/m |
| Ground normal damper constant | 250 Ns/m |
| Ground static friction coefficient | 2.5 |
| Ground kinetic friction coefficient | 1.8 |

**Table 7.1**: *Some environmental constants for the biped robot simulation.*

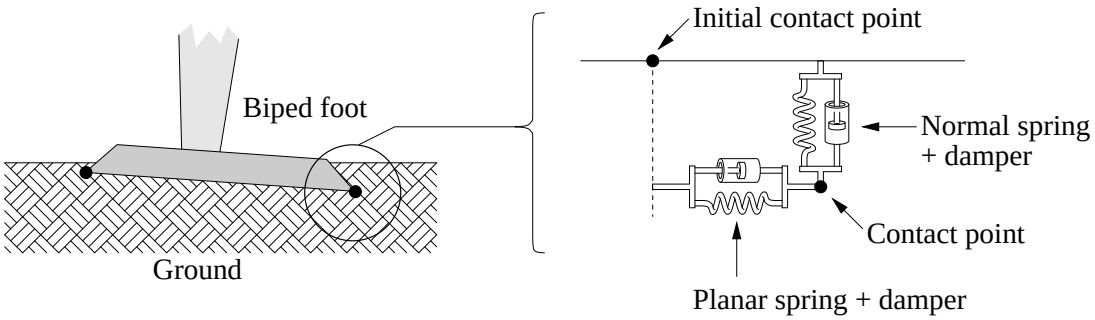


**Figure 7.1**: *Ground contact forces are modeled as two spring+damper units between the point of first contact and a collision detection point on the solid. Forces normal and planar to the ground are handled separately.*

  - A dynamic network system description language called 'DND'.
  - A compiler called '`dnd2exe`' (written in Perl) to convert DND into executable C code.
  - A simulation kernel (written in C) to simulate the system.

  See Appendix J for further details.

- `CyberSim` performs the actual robot simulation, it brings together `RoboDyn` with the output of `Dyson`. See Appendix H for further details.
- `CyberView` allows the user to move through a virtual 3D environment and observe the operation of the robot over time (the output of `CyberSim`). The custom graphics engine (the `R3D` library) supports flat shaded rendering to an X-terminal or texture mapped real-time rendering to a 3dfx/Voodoo graphics accelerator card. See Appendix H for further details.

In the virtual environment, the ground was modeled as a spring-plus-damper system with independent coefficients for normal and planar sliding contact (see figure 7.1). Some of the ground and environment parameters are given in table 7.1. Both viscous and Coulomb friction were modeled, and both static and sliding contact modes were supported, each with different coefficients. Every robot joint had built in spring-plus-damping limits on its motion, as well as internal viscous and Coulomb friction.

However, the mechanical models were unrealistic in several respects. Ideal torque-motors were used instead of more realistic (and dynamically complex) electromagnetic motor models. There was no slip-

page between the actuators (motors) and joints. The joint sensors were assumed to have unlimited resolution and be free from noise. And finally, no collision detection was performed between the robot links. This is most apparent in the biped experiments, where the biped's feet occasionally move through each other.

Hence despite the sophistication of the simulation, it is definitely not as good as a having a real robot when it comes to proving a new control methodology. However, real robots were not constructed for two reasons. First, they are obviously expensive and difficult to build. Second, they hinder a trial-and-error approach to controller construction and learning, because the mundane details of performing experiments on them are extremely time consuming when compared to simulation. This is not an argument for abandoning hardware, just a statement that simulation is particularly useful when new and relatively untried methods are being tested.

## 7.5 Hopping robot

### 7.5.1 Introduction

A simulated single legged hopping robot was created to provide a simple test of the above controller design strategy. Figure 7.2 shows the hopping robot in its virtual environment. The robot is roughly one meter high and has a 10kg body supported on a telescopic leg. An actuator can apply a force to the leg to make it extend or contract. "Hip" actuators are able to rotate the leg around a ball and socket joint that attaches it to the body. The robot's sensors measure leg and hip position, foot contact with the ground, the body speed along the x and y axes (parallel to the ground), and the slope of the ground under the foot (along the x and y directions). The robot's task is to follow an approximately square trajectory along the ground, which takes it up and down a ramp.

### 7.5.2 The basic controller

An outline of the robot's controller is shown in figure 7.3. It is similar in principle to the hopping robot controllers described by Raibert [103] and Boone [17]. The control problem is decoupled in to three relatively independent problems: control of jumping height, control of body angle while the foot in on the ground, and control of leg position when the foot strikes the ground (to indirectly achieve speed control).

All actuators are controlled using 'LLJC' (low level joint controller) modules which implement simple proportional plus derivative control:

$$\text{force} \quad = \quad \text{gain} \cdot (\text{ref} - \text{pos}) - 100 \cdot \frac{\mathrm{d}}{\mathrm{d}t}(\text{pos}) \tag{7.1}$$

Where 'pos' means position and 'ref' means reference. For the hopping robot the gain is 1000 unless otherwise specified. A simple state machine selects the gain of the LLJC that controls the leg length, to achieve a hopping motion (state 1 selects the high gain which thrusts the leg out).

There are two hip actuators to control the x and y rotation of the leg to allow hopping in any direction. For simplicity, only the controller for one axis is shown in figure 7.3. When the foot is on the ground an LLJC controls the hip actuator to try and achieve a zero body angle $\phi_b$. This would eventually cause the robot to tip over if the foot stayed on the ground. When the foot is off the ground another LLJC controls the leg angle relative to the body. The reference is determined from the following control rule, which tries to maintain the forward speed of the body at some reference value:

$$\text{ref} \quad = \quad k_1 \cdot \text{speed} - k2\ \phi_b - k_3\ (\text{desired\_speed} - \text{speed}) - k_4\ \text{slope} \tag{7.2}$$

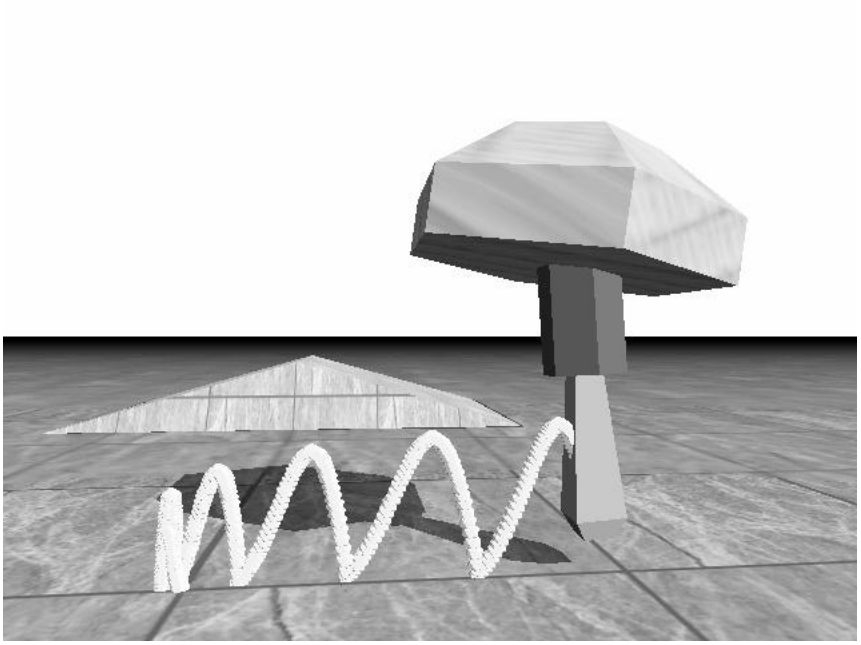

**Figure 7.2**: *The monoped hopping robot in its virtual environment. As the foot moves it leaves a trail behind, which is useful for showing the robot path in these still images. In the background is a ramp which the robot must climb and descend.*

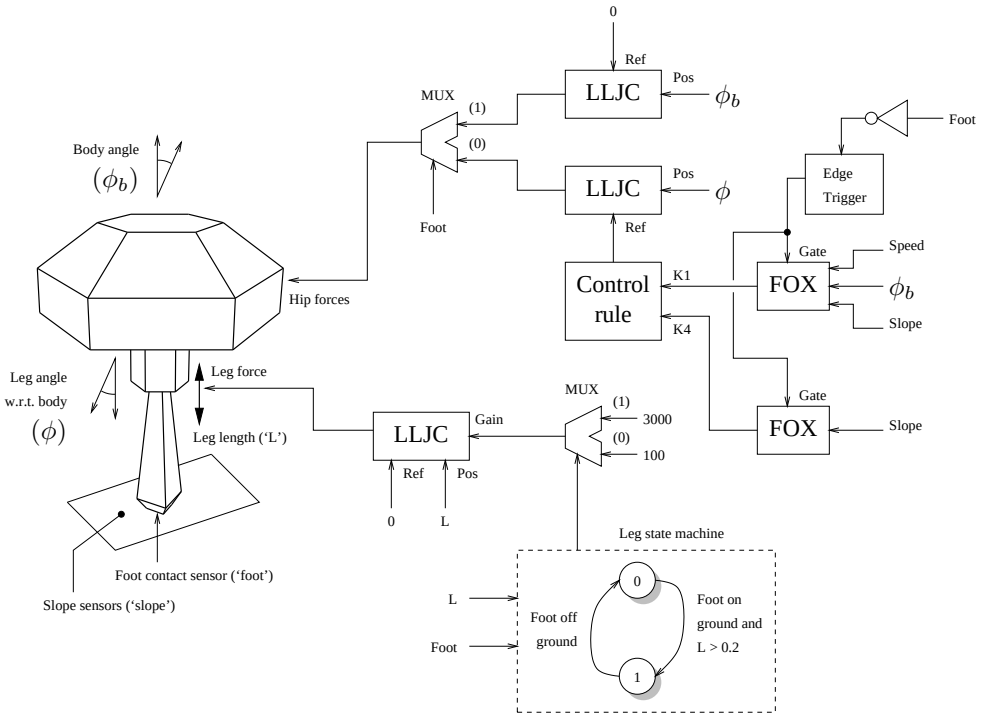


**Figure 7.3**: *A simplified view of the hopping robot's controller. Many details are omitted from this figure.*

where $k_1 \ldots k_4$ are constants. The $k_1$ term is the main foot positioning gain, to maintain the current speed. The $k_2$ term helps compensate for the body tilt when placing the foot. The $k_3$ term corrects speed errors, and the $k_4$ term adjusts the leg angle on a slope for better balance (this is necessary because the foot has a significant width).

### 7.5.3 The learning controller

Selection of the four control parameters to achieve stable control is tricky, but it is certainly possible, either from experiment or from an analytical understanding of the robot's motion. However, the task here is to use FOX to determine some of them from experience and (in principle at least) save the designer from finding them. This has other advantages: the controller is potentially better adapted to this nonlinear system than a pure linear controller (for example the optimum value of $k_1$ varies with speed), and the controller is more context sensitive, that is better adapted to different environments (different ground slopes in this case). Figure 7.3 shows how two FOX modules provide $k_1$ and $k_4$ (good values for $k_2$ and $k_3$ were found experimentally to be $0.8$ and $0.1$).

The $k_1$ FOX used overshoot training with output limiting. Its error signal was:

$$e \quad = \quad -(\text{desired\_speed} - \text{speed}) \cdot \text{sgn}(\text{speed}) \tag{7.3}$$

The sgn$(\cdot)$ functions returns $\pm 1$ depending on the sign of its argument and is used to ensure that $k_1$ will be positive for motion in both the positive and negative directions. Overshoot training is essential in this case to prevent $k_1$ increasing without bound whenever the desired speed is changed. The output of the FOX is sampled and held stationary each time the foot leaves the ground. This makes the control problem slightly more difficult (the FOX cannot correct $k_1$ during the flight of the foot) but it prevents the leg from undergoing an oscillatory "hunting" behavior where it tries to seek the correct position (as the value of $k_1$ varies) during foot flight. The FOX is gated by an edge triggered version of the foot signal, that is the gate is only turned on at the instant when the foot leaves the ground (see section 5.9.2 for a definition of the gate). This is essential for training because the FOX output can only affect the system at the gated times. A critically damped eligibility profile is used with $t_{\text{max}} = 1$ (refer to Appendix E for the definition of $t_{\text{max}}$).

The $k_4$ FOX is configured similarly, except that its error signal is designed to only provide correction along slopes:

$$e \quad = \quad -(\text{desired\_speed} - \text{speed}) \cdot \text{sign}(\text{speed}) \cdot \text{slope} \tag{7.4}$$

### 7.5.4 Results

In the early stages of training the robot has not learned to correct its speed when it goes too fast, and so it trips and falls over frequently (figure 7.4). A movie of four out-takes from training was created (see movie `hopper1.mpg`) which shows the robot falling over in progressively later stages of its desired trajectory (right at the start, turning the first corner, going up the ramp, and going down the ramp). It shows that the robot must learn independently how to move in all four compass directions as well as up and down slopes.

After 20 training iterations (each iteration corresponds to falling over once) the robot has acceptable performance. Figure 7.5 shows four images from one attempt to follow the trajectory (also see movie `hopper2.mpg`).

The robot can hop along at a constant speed on the flat, it can turn corners, and climb and descend the ramp without falling. Its descent down the ramp is rather too fast, and in fact the controller described

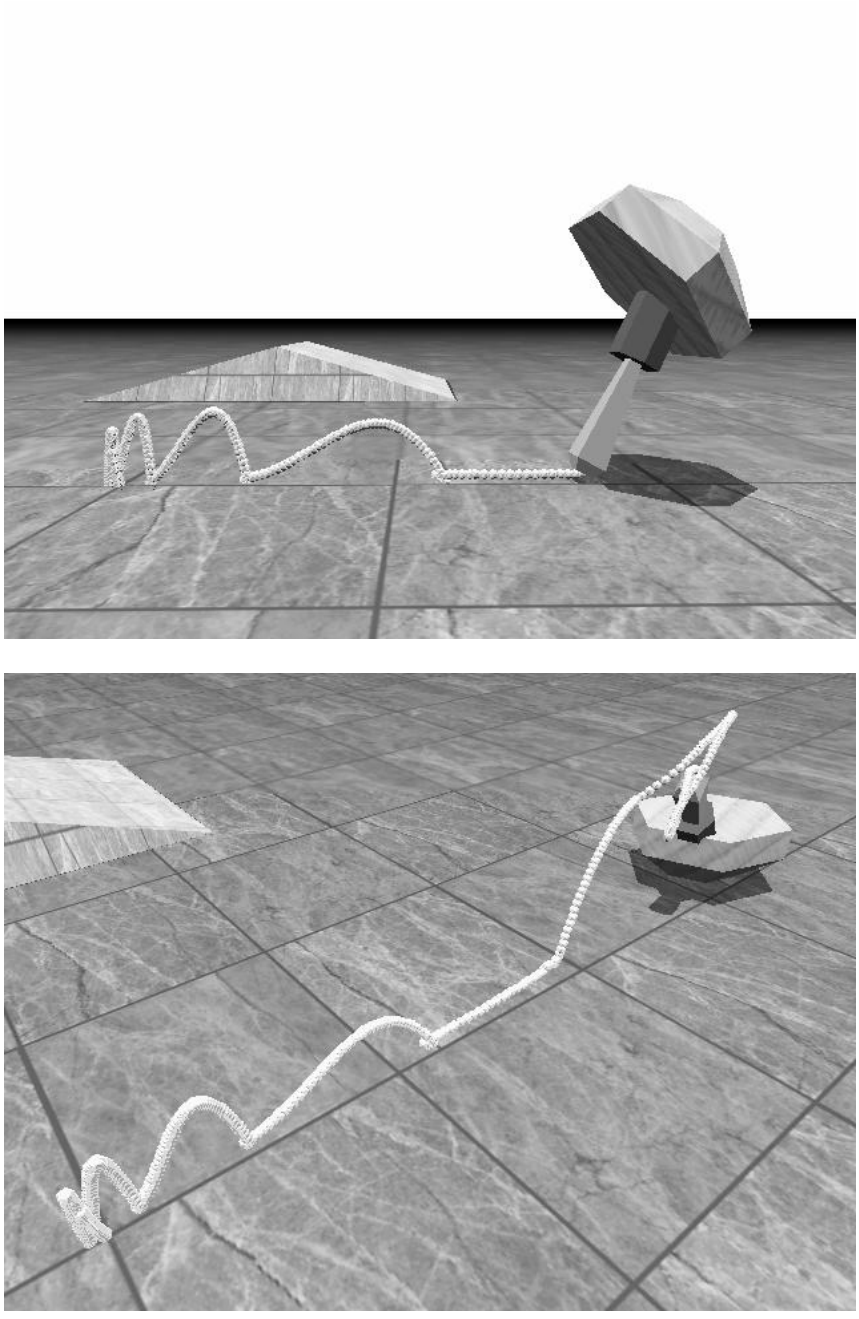

**Figure 7.4**: *The monoped hopping robot in the early stages of training. The robot has not learned to correct its speed when it goes too fast, so as a result it trips and falls over.*

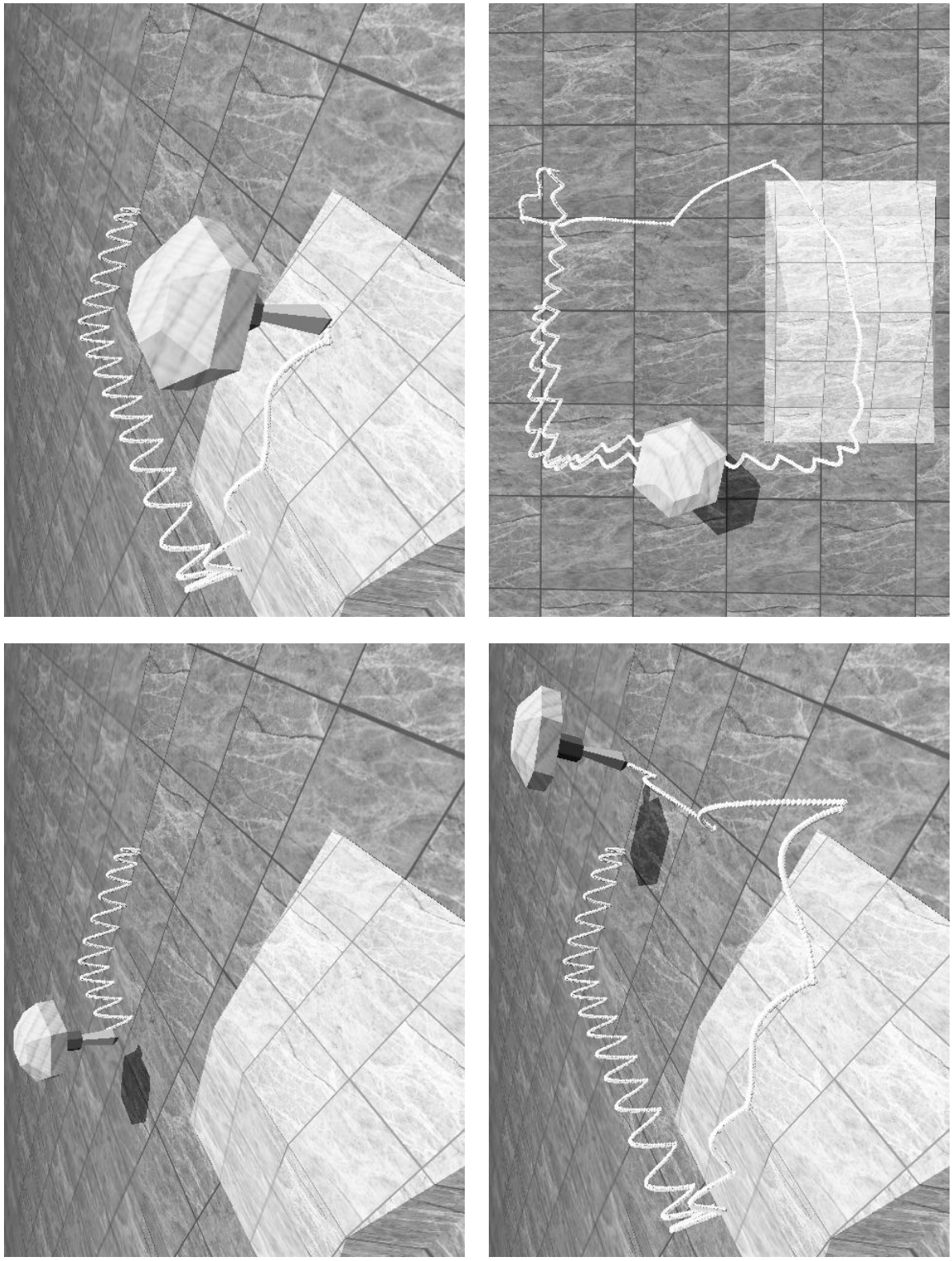

**Figure 7.5**: *The monoped hopping robot after 20 iterations of training (each iteration corresponds to falling over once). The robot follows a roughly square path that takes it up and down the ramp. The images sequence is bottom left, top left, bottom right, top right.*

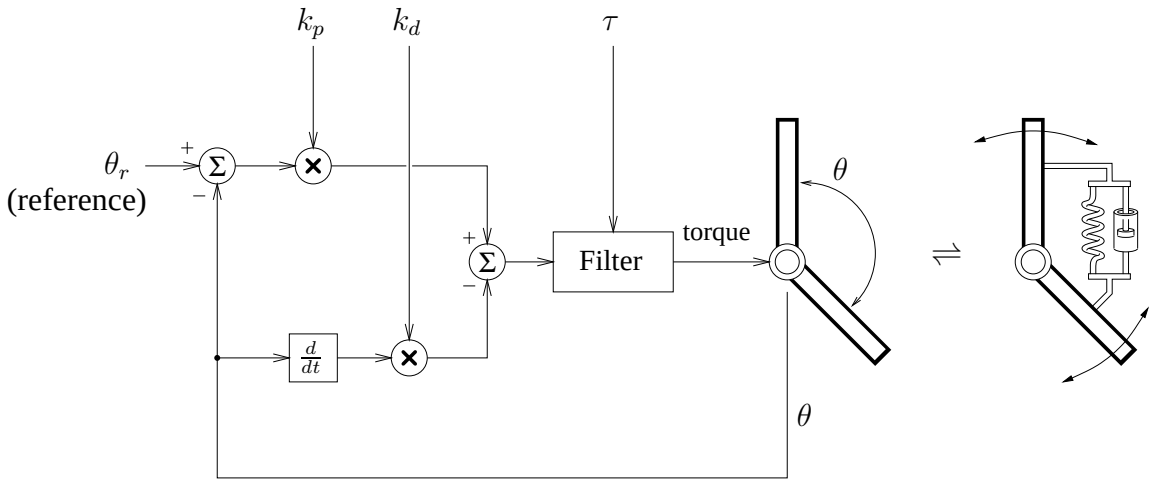


**Figure 7.6**: *The low level joint controller (LLJC), which is equivalent to having a spring-and-damper on the joint with an adjustable spring set-point.*

here never learns to limit the down-slope speed. This is an important point: the FOX modules do not have the freedom to optimize the entire motion of the robot. Instead they are just allowed to control two parameters of a controller with a fixed complicated structure. The FOX modules are only capable of changing the robot's behavior in limited ways. In general this can be beneficial, in the sense that training can never override behaviors that the designer has already determined to be useful. In other words, the trade off is between a complicated controller parameterized by FOXs with simple error functions, or a simple controller parameterized by FOXs with complicated error functions.

The hopping robot experiment provided a useful starting point for designing the more complicated biped controller. It introduced the idea of control with state machines and using different control modules in different parts of the robot's trajectory. It has also demonstrated that learning controller parameters is an effective alternative to explicitly learning an entire control force profile.

## 7.6 System components

Now a more detailed look will be taken at the components that can be used in robot controllers. The controllers for the hopping and walking robots are combined continuous and discrete dynamical systems. Internal state is maintained by discrete state machines, or timers, or less frequently by integrators. Different modules implement different behavioral tasks, and the modules pass information to each other to coordinate their activities.

At the lowest level, most actuators are controlled using 'LLJC' (low level joint controller) modules which implement simple proportional plus derivative joint-position control. The controller of figure 7.6 is used, which provides a simple model of the muscles and the most important spinal cord reflexes, in accordance with the conclusions of Latash [65]. It is equivalent to having a spring-and-damper on the joint with an adjustable spring set-point.

Instead of a single global performance error there are a number of local error values that each apply to a small piece of the problem. For example, errors may be formulated to keep the robot body level when one leg is on the ground, or to make sure the other leg is lifted far enough to clear any obstacles

when the controller is in a certain state.

Many types of controller parameters can be learned by FOX modules. For example:

- Coefficients for LLJC modules.
- Desired joint positions at various stages of locomotion, or the end positions for movements.
- The timing of state changes.
- The selection of state changes, that is, the potentiation of the next state that will be entered.

The last item (state potentiation) has been shown to be useful in the author's previous work [115] for high level behavioral selection. FOX modules can co-operate to control a single parameter (if there are multiple constraints on a single value), but in many instances the FOX modules act independently. For this to work well, the variables controlled by each FOX must be as independent as possible, so that minimization in one variable does not affect the others. In practice this is rarely possible, so there is often a lot of undesired interaction between the various learning processes. This can make the system's performance worse in the short term, but in a well designed system things will usually settle down and performance will improve.

The controllable parameters will differ in the level of influence they have on the system. Low level control parameters will contribute directly to the actuator outputs (the joint forces). Higher level control parameters will undergo additional processing by the controller (for example, the joint position references). It is generally preferable to control high level parameters, because the additional controller processing can generally tailor the system's nonlinear dynamics so that the eligibility profile concept is more applicable (this may be more difficult if low level parameters like force are being controlled). In general, nonlinear state machine based controllers stretch the FOX theoretical model to its limit—in many cases it will just be assumed that FOX control will work, without any rigorous justification. However, it is anticipated that additional "internal" feedback controllers will usually be incorporated into the system to provide appropriately modelable dynamics.

Another reason for FOX modules to control high level parameters is that this can result in far more robust control. This is because when the system enters an unforeseen state (one previously untrained) and the FOX outputs are zero, the controller still has a chance of providing an adequate response if the parameter value of zero results in some default behavior. If the FOX controls a parameter that is too low level (such as position or force) then a zero output will be inappropriate in almost every situation, and system failure during learning will be much more frequent.

Many parameters will only influence the system for part of the time (the rest of the time the system will be in a state where the parameter is not even considered). In these cases the FOX eligibility driving force must be gated, as explained in section 5.9.2. If this does not happen then the FOX may learn inappropriate things. However, an alternative in some situations is to gate the FOX error signal instead.

The FOX error signals, error function and learning rates have to be chosen with care, and some experimentation is usually needed before a working combination can be found. The FOX eligibility profile may be difficult to create for a highly nonlinear robot-plus-controller system. But in most cases it has been found that a second order critically damped profile with an estimated $t_{\max}$ value in the range $0.1\text{s} \ldots 2\text{s}$ is sufficient (see Appendix E for the definition of $t_{\max}$).

A design compromise must be made when choosing the inputs to each FOX. Fewer inputs mean that the FOX will be able to generalize its training to a wider range of unforeseen situations. But more inputs mean that the FOX will be able to individually tailor its output to a wider range of specific situations.

If the controller had no internal state, that is if it was purely sensor driven, then a given sensor picture would always result in the same actions. It is usually more useful to have one or more state

machines or timers inside the controller so that it can have an internal notion of its current "plan" which is independent of the outside world. Transitions between states can be triggered when some function of the sensors climbs above a threshold, or when a time limit for the current state has been reached.

When designing a controller a choice must be made between state-dependent (feedback) control and state independent (feed-forward) control. A feedback controller depends completely on its sensors to determine state changing decisions and actuator outputs. A feed-forward controller makes state changing and actuator output decisions according to some pre-arranged schedule. In practice both methods are used together, although a particular controller may be biased towards one or the other. Feed-forward control allows a pre-arranged behavior to be played out—some parameters of this behavior are learned so that it will be correctly mapped to the environment. But feed-forward control relies on the system being in a particular known state (or limit cycle) and it will not work well otherwise. With feedback control it can be trickier to get a particular sequence of movements, but the controller is usually a lot more robust to body states outside those anticipated by the designer. The hopping robot was largely feedback-based. The biped robot which will be described in the next section has feed-forward components.

## 7.7 Making a biped robot walk

### 7.7.1 A short review of biped locomotion

The problem of biped robot locomotion has received a lot of attention. This is partly due to the great difficulty of the problem (and therefore its high research value), partly because of the desire to understand the principles behind human locomotion, and partly due to the presumed superiority of bipeds over quadrupeds and hexapods, and wheeled or tracked vehicles (after all, human walking is highly adaptive and versatile).

Despite this, the problem is largely unsolved. Although there are now many real and simulated bipeds that can walk and run, none of these systems can survive for long in unstructured environments without falling down. Although more than 60 different climbing and walking machines developed in research laboratories and universities have been cataloged, industrial applications are emerging very slowly [106]. No *bipeds* have yet been deployed in any practical applications, although various robot with four or more legs have been.

Legged robots are either statically or dynamically stable [18]. Statically stable robots will retain their balance if they stop moving at any time, thus they present a relatively easy control problem. They typically have four or six legs but may be bipeds with large feet. Dynamically stable robots are only stable in a limit cycle that repeats once each stride. They are much more difficult to control, but (in theory at least) can provide more versatile locomotion.

From the point of view of current control theory the dynamically stable locomotion problem is not overwhelming. Indeed, several analytical studies have been made which show how biped walking can be achieved. For example, Kajita and Tani have studied dynamic biped walking using a "linear inverted pendulum" model for leg movement. Their scheme uses linear control approximations, and has been tested on a real six degree-of-freedom robot [56]. Similarly, Pannu *et al* have tested the analytical $\mu$-synthesis control approach on one leg of a walking robot [93].

Raibert and his colleagues have built dynamically stable running monopeds, bipeds, and quadrupeds [18]. These robots all achieved locomotion by bouncing on springy legs. Their control systems achieved stability by decoupling the problem into three parts: (1) controlling hopping height through the leg actuators, (2) controlling forward speed by correctly positioning the foot at touchdown, and (3) controlling the body attitude by applying torque at the hip while the foot was in contact with the ground [103]. Finite state

machines were used to switch between the different control laws required in each mode. Three dimensional hopping and running was achieved by decomposing the robot motion into planar and extra-planar parts [104]. Recently this work has been extended to make the locomotion more robust in real-world environments: Boone has implemented reflex-like actions to help biped robots recover from slipping and tripping [17].

Almost all current implementations rely on a well structured environment (usually just a hard, flat walking surface). Most current implementations can not cope with uneven terrain, slopes, varying contact friction, and obstacles—that is, they are not very adaptive. This is because most walking machines to date do not learn: their control parameters are computed analytically or adjusted by hand. But there are a few exceptions. Miller has achieved balanced walking in a real ten axis biped robot using CMACs to learn some controller parameters [83, 84]. The controller contains a hierarchy of gait oscillators, PID controllers, and CMAC networks. His scheme does not require a detailed robot dynamical model, but the robot can only take short steps without falling over. Lin and Song have also used CMAC neural networks for the purely kinematic control of legged robots [70]. Stitt and Zheng have developed a method that generates biped gaits suited to varying ground slopes, based on "distal supervised learning". Their technique requires a forward model of the robot's dynamics, and converts information about stable gait on a flat floor to rules for climbing and descending slopes [116].

Many controllers have been designed using semi-biological principles. For example, Crawford has designed a hierarchical control structure that uses radial basis function networks, for the control of a human platform diver [30, 29]. It is proposed that systems with many degrees of freedom can be controlled with a hierarchical network of simple single-joint learning controllers. Hallam *et al* go even further with their neuroethological approach in which a neural network with quasi-realistic synapse modification is used to control a robot [41].

The difficulty of the biped locomotion problem makes it a good test bed for new control theories. Failure in biped locomotion is generally catastrophic, as no current system has the coordination to pick itself up once it has fallen down. Thus one can always gauge the success of biped control schemes in an ad-hoc manner by asking "how long can it walk without falling over?".

The biped controller presented here is more adaptive than most, but it is still unsuited to any practical application. The chief value of these experiments is to test the value of FOX-based controllers in complex robot problems.

### 7.7.2 The problem

Experiments were performed on the simulated biped robot shown in figure 7.7. The biped has one significant mechanical deficiency compared to a human—it has no lower back / hip flexibility. The biped's mass parameters are given in table 7.2.

The robot was given three tasks:

1. Learn to walk in a straight line with a steady gait.
2. Learn to walk in a circle (to test the robot's ability to change direction).
3. Learn to walk up and down a ramp (to test the robot's ability to adjust its gait to the slope of the ground).

### 7.7.3 Feed-forward controller structure

A feed-forward walking approach similar to Laszlo *et al* [64] was used. In that study a walking biped (with 19 degrees of freedom) was simulated using limit cycle control, with the intention of rendering

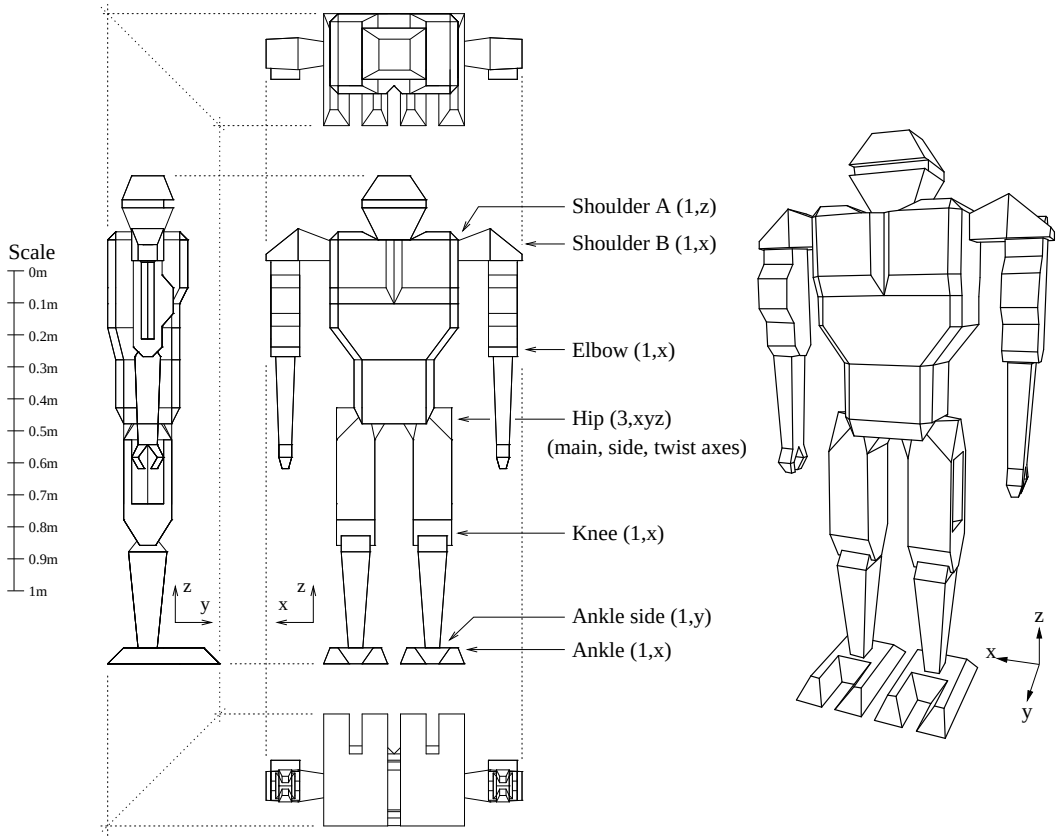


**Figure 7.7**: *The structure of the simulated biped robot. The robot has 18 degrees of freedom spread over seven joints on each side of the body. Each joint is labeled with the number of degrees of freedom it has and the axes (x,y,z) it can rotate through. The hip joint is a ball-and-socket which is controlled along the main (x), side (y) and twist (z) axes. The other joints are simple revolute (hinge) joints.*

| Link | Mass (kg) | Rotational inertia (kg m$^2$) | | |
|---|---|---|---|---|
| | | $x$-axis | $y$-axis | $z$-axis |
| body | 30.0 | 0.9125 | 1.15625 | 0.1696 |
| upper leg | 4.0 | 0.2208 | 0.2181 | 0.0123 |
| lower leg | 2.5 | 0.0047 | 0.1038 | 0.1051 |
| foot | 0.5 | 0.08 | 0.018 | 0.065 |
| shoulder link | 1.0 | 0.014 | 0.0017 | 0.014 |
| upper arm | 2.0 | 0.0014 | 0.083 | 0.083 |
| lower arm | 1.5 | 0.0009 | 0.0617 | 0.0617 |

**Table 7.2**: *The mass parameters of the simulated biped robot.*

realistic looking walking for computer graphics applications. Stereotyped open-loop (feed forward) periodic walking motions were generated using a finite state machine (they were not strictly open-loop as PD controllers were used at each joint, but there was no global feedback to hold the system on its desired trajectory). By themselves these motions did not result in stable walking, as sensor feedback was not used. The walking motion was stabilized by adding closed loop feedback. An off-line hill-climbing learning method was used to find the control parameters that kept the walking motions on a limit cycle (so that falling-over perturbations were automatically corrected for). The biped's direction, speed and stride rate were controllable.

Figure 7.8 shows the internal structure of the controller used in this experiment. Each of 18 joints (six in each leg and three in each arm) are controlled by an LLJC module. A variety of high-level modules control various aspects of the biped's motion. Figure 7.9 shows a stick-figure representation of how each controller module contributes to the biped's motion. Each module produces desired reference values for one or more joints. These references are combined together in the RC (reference computation) units which compute the final joint references (see table 7.4 for details, and table 7.3 which defines the symbols used). Eighteen unknown parameters are learned by FOX modules (eight which are duplicated in each leg and two which are global—see table 7.5). Second order critically damped eligibility profiles are used (defined by the parameter $t_{\max}$, see Appendix E). The values of $t_{\max}$ were selected somewhat arbitrarily, based on a mixture of intuition, guesswork and experimentation.

Note that there a *many* possible controller structures, and also a great number of potentially learnable parameters. The selection used in this experiment is not claimed to be the best: the purpose of the experiment is merely to demonstrate FOXs usefulness in this kind of system.

The operation of the biped after training is shown in figure 7.10, figure 7.11, and in movie `walkt2.mpg`. The robot can successfully walk with a steady gait, change its stride length, change direction and climb and descend slopes.

The controller operation will now be described (note that many details have been omitted for the sake of clarity). It will be shown how each controller module separately influences the behavior of the system, by showing what happens when that controller is removed.

A variety of sensors are available to the controller: 18 joint angular positions and velocities, the absolute body orientation along three axes ($\phi, \theta, \psi$), contact forces at the heel and toe, and the slope a short distance in front of the robot. A simple state machine for each foot (not shown) determines when a solid contact has been made with the ground.

The principle goal of the controller is to keep the body upright while maintaining the desired forward speed and zero sideways deviation from the desired path. The basic motions of walking are generated by

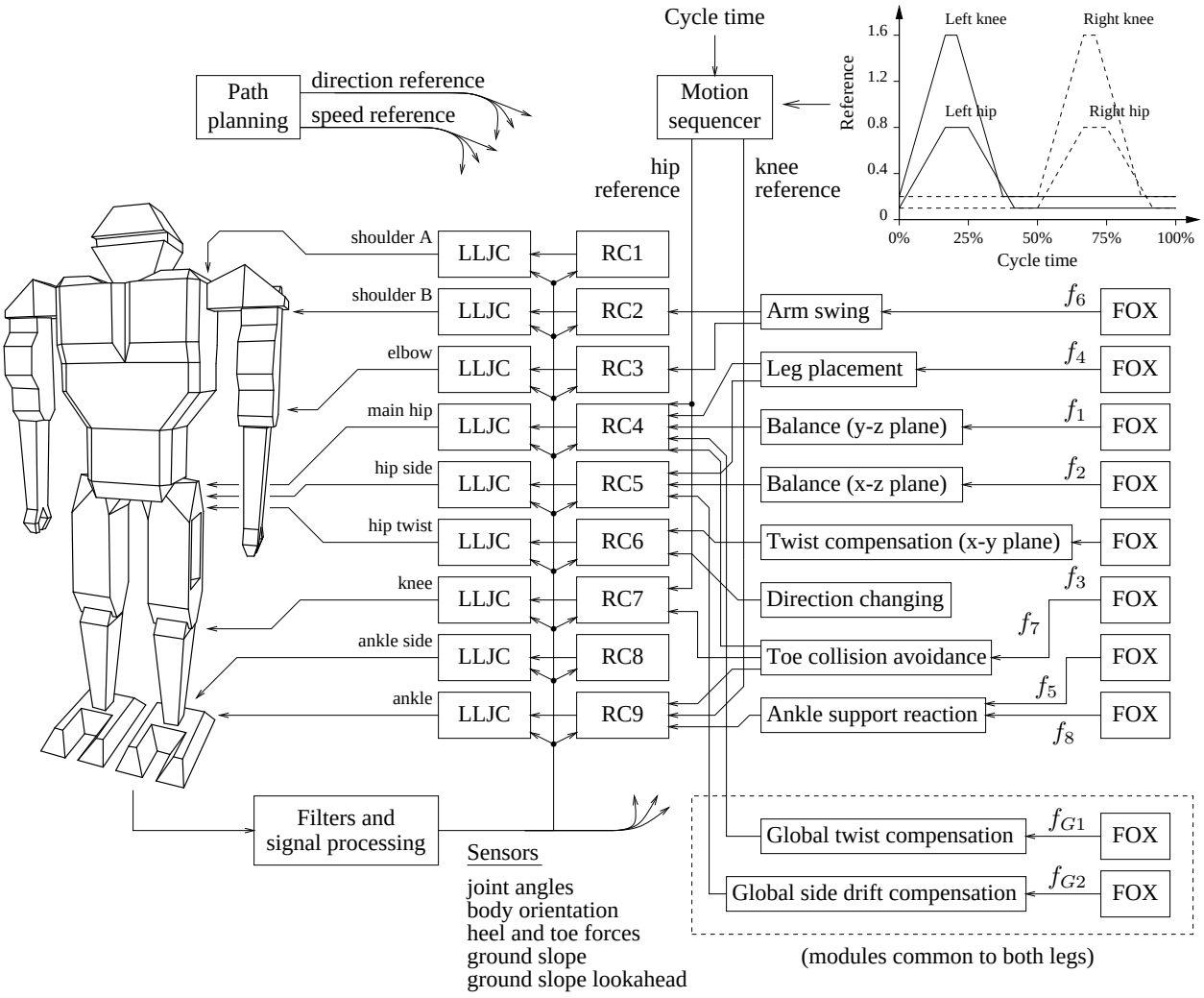


**Figure 7.8**: *The structure of the "feed-forward" controller for biped walking. Note that many details are omitted from this diagram.*

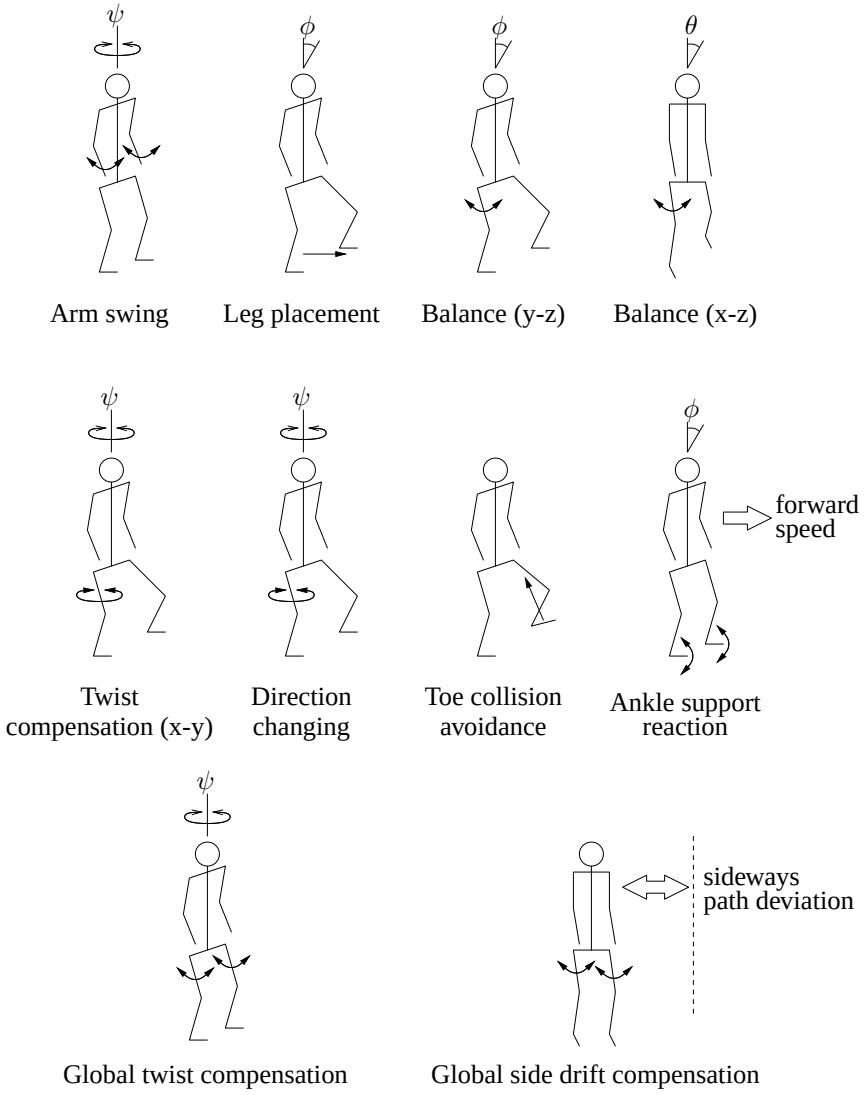


**Figure 7.9**: *A stick-figure representation of how each controller module contributes to the biped's motion. In each figure the arrows show the controlled joint, and the corresponding error variable is indicated (either $\phi$, $\psi$, $\theta$, forward speed or sideways deviation).*

| Symbol | Definition |
|---|---|
| | Motion sequencer |
| $R_1$ | Hip reference |
| $R_2$ | Knee reference |
| | Desired values |
| $r_1$ | Desired value of $\phi$ (forward tilt) |
| $r_2$ | Desired forward speed |
| $r_3$ | Desired value of $\psi$ (direction) |
| | Errors |
| $e_1$ | $\phi$ error $= r_1 - \phi$ |
| $e_2$ | speed error $= r_2 - s_3$ |
| $e_3$ | sideways path deviation $= -s_6$ |
| | Sensors |
| $s_1$ | Is foot touching the ground? (boolean) |
| $s_2$ | Is the opposite foot touching the ground? (boolean) |
| $s_3$ | Forward speed (m/s) |
| $s_4$ | Is the foot firmly on the ground? (boolean) |
| $s_5$ | Is the opposite foot firmly on the ground? (boolean) |
| $s_6$ | Sideways path deviation (m/s) |
| $s_7$ | Contact force at heel (N) |
| $s_8$ | Contact force at toe (N) |
| $s_9$ | Slope of ground just in front of robot (radians) |
| $\phi$ | Body angle in the y-z plane (forward tilt) |
| $\theta$ | Body angle in the x-z plane (sideways tilt) |
| $\psi$ | Body angle in the x-y plane (twist) |
| | FOX outputs |
| $f_1$ | Reference adjustment for main hip |
| $f_2$ | Reference adjustment for hip side |
| $f_3$ | Reference adjustment for hip twist |
| $f_4$ | Reference adjustment for main hip (2) |
| $f_5$ | Reference adjustment for ankle |
| $f_6$ | Reference adjustment for shoulder B |
| $f_7$ | Reference adjustment for hip,knee and ankle |
| $f_8$ | Reference adjustment for ankle |
| $f_{G1}$ | Global reference adjustment for main hip (3) |
| $f_{G2}$ | Global reference adjustment for hip side (2) |
| | Miscellaneous |
| sign | $+1$ on the left, $-1$ on the right |
| $t_c$ | cycle time $(0 \ldots 1)$ |
| $a_s$ | action sine $(-1 \ldots 1)$ |
| $a_c$ | action cosine $(-1 \ldots 1)$ |

**Table 7.3**: *Definitions of the symbols used in the walking biped controller (see table 7.4). The values $f_1 \ldots f_8$ and 'sign' are duplicated for each leg.*

| Joint | Reference (radians) |
|---|---|
| Shoulder A | 0 |
| Shoulder B | $-R_1(0.7 + f_6) + 0.15$ |
| Elbow | $0.5 + 0.5R_1$ |
| Hip (main) | $R_1 - r_1 + (\text{not } s_1)(1.5e_1 + 0.8s_3) + f_1 s_4 + \text{sign } f_{G1}(s_1 \text{ and } s_2) + (\text{not } s_1)(f_4 + f_7)$ |
| Hip (side) | $f_2(s_4 \text{ and not } s_5) + f_{G2}(s_1 \text{ and } s_2) + (\text{not } s_1)(-0.1 \text{ sign} - 2\theta - 0.3s_6)$ |
| Hip (twist) | untwist_ref $+ f_3 s_4$ |
| Knee | $R_2 - 2(\text{not } s_1) f_7$ |
| Ankle (side) | $2\theta + 0.5\dot{\theta} + 0.3e_3$ |
| Ankle | $(s_1 \text{ and } R_1 \leq 0.15)((-0.12 - 0.003(s7 - s8)) + 0.3e_1) - (\text{not } s_1 \text{ or } R_1 > 0.15)R_1 - (\text{not } s_1) f_7 + s_4(f_5 + f_8)$ |

**Table 7.4**: *The joint references computed by the RC modules for the walking biped (incorporating all controller modules and FOX-learned parameters). Note that some details are omitted. For definitions of the symbols used, see table 7.3.*

| FOX | Inputs | Gate | Error | $t_{\max}$ (s) | Learning rates |
|---|---|---|---|---|---|
| $f_1$ | $a_s, a_c, s_3$ | $s_4$ | $-e_1$ | 0.4 | $\alpha_2 = 0.0005, \beta = 0$ |
| $f_2$ | $a_s, a_c, s_3$ | $s_4$ and not $s_5$ | $\theta$ | 0.2 | $\alpha_2 = 0.005, \beta = 0.00001$ |
| $f_3$ | $a_s, a_c, s_3$ | $s_4$ | $\dot{\phi} s_4$ | 0.4 | $\alpha_2 = 0.005, \beta = 0.00001$ |
| $f_4$ | $a_s, a_c, s_3, s_9$ | not $s_1$ | $e_1$ | 0.5 | $\alpha_2 = 0.0002, \beta = 0$ |
| $f_5$ | $a_s, a_c, s_3$ | $s_1$ and $R_1 \leq 0.15$ | $e_2$ | 0.05 | $\alpha_2 = 0.00005, \beta = 5 \times 10^{-7}$ |
| $f_6$ | $a_s, a_c, s_3$ | 1 | $\dot{R}_1 \dot{\psi}$ | 0.2 | $\alpha_2 = 0.0005, \beta = 0$ |
| $f_7$ | $a_s, a_c, s_3$ | not $s_1$ | $s_1$ and $t_c < 0.3$ | 0.3 | $\alpha_2 = 0.05, \beta = 0.005$ |
| $f_8$ | $a_s, a_c, s_3, s_9$ | $s_1$ and $R_1 \leq 0.15$ | $e_1$ | 0.5 | $\alpha_2 = 0.0005, \beta = 5 \times 10^{-6}$ |
| $f_{G1}$ | $a_s, a_c, s_3$ | $s_1$ and $s_2$ | $\dot{\psi}(s_1 \text{ and } s_2)$ | 0.2 | $\alpha_2 = 0.001, \beta = 0.00001$ |
| $f_{G2}$ | $a_s, a_c, s_3$ | $s_4$ and $s_5$ | $-e_3$ | 1.0 | $\alpha_2 = 0.01, \beta = 0.001$ |

**Table 7.5**: *The inputs and parameters of the FOX modules used in the walking biped controller. In all cases, overshoot learning with output limiting is used, with $\alpha_1 = 0.01\alpha_2$. Second order critically damped eligibility profiles are used, defined by the parameter $t_{max}$ (see Appendix E).*

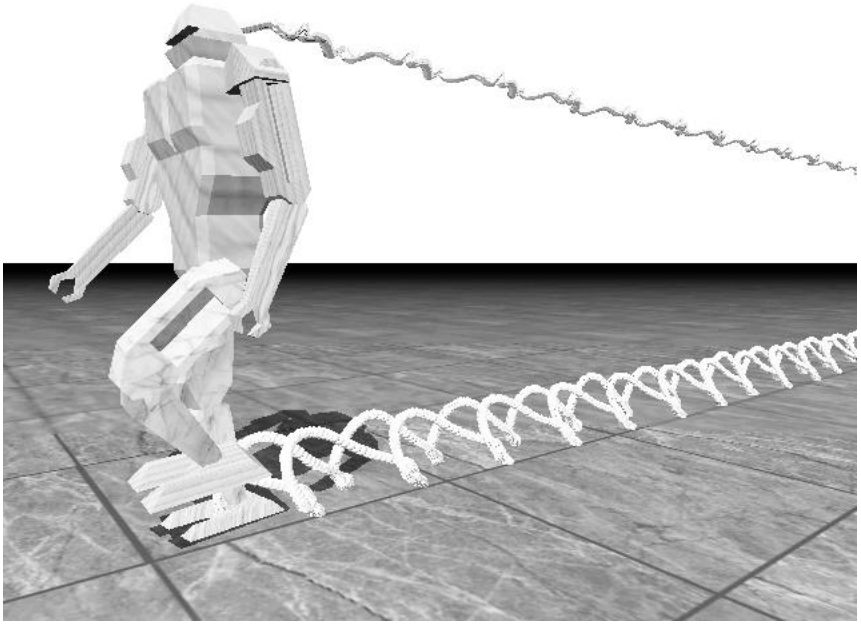

**Figure 7.10**: *Successful biped walking. Note that trails are rendered from the top of the head and from both ankle joints.*

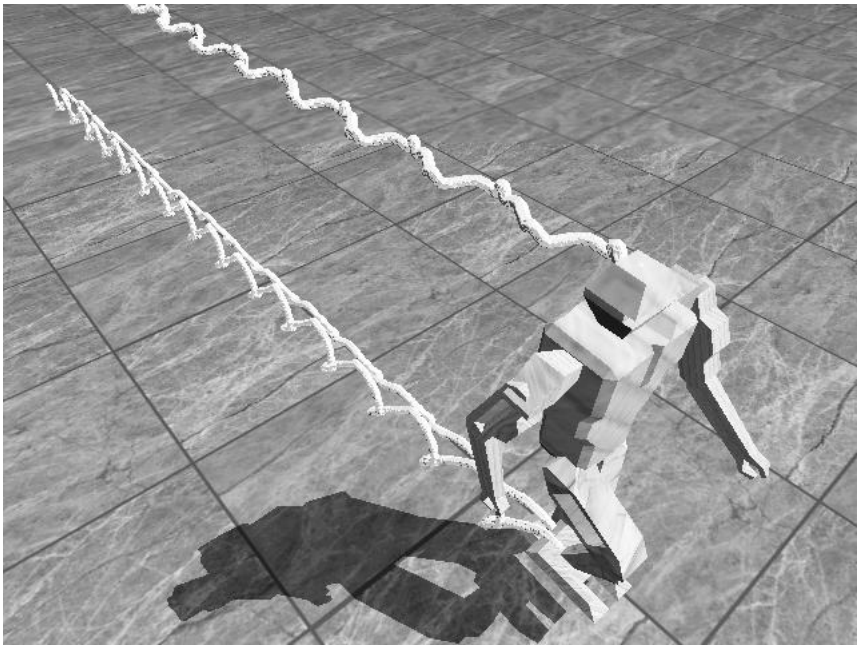

**Figure 7.11**: *Successful biped walking—another view.*

the motion sequencer which outputs periodic preprogrammed hip,knee and ankle joint references (at a fixed frequency of 0.75Hz) which extend and contract the leg. Without the other controller modules the motion sequencer can drive the robot through only a couple of steps before it falls over (see figure 7.13, movie `walk1.mpg`).

All FOX modules used input resolutions between 80 and 200, and an $n_a$ of 20. Each of the 18 FOX modules had 100,000 weights (far more than were really needed), for a total of 1.8 million weights in the entire system.

To achieve stable limit-cycle walking it is expected that the FOX outputs would have to vary in accordance with the cycle time. Thus the FOX inputs all include the variables $a_s$ and $a_c$. These variables are sine and cosine functions of the cycle time: the point $(a_c, a_s)$ goes around the unit circle once per cycle. They are a used instead of the cycle time itself to prevent CMAC local generalization discontinuities once per cycle. One advantage of the feed-forward mode is that the FOX modules need few other inputs to achieve sufficient specialization.

During the walking cycle each leg sees three different situations: (1) the foot is in the air, (2) the foot is on the ground while the other foot is in the air, and (3) both feet are on the ground. Each controller module is normally switched in during only one or two of these situations.

**y-z and x-z balancing controllers:** These balance the robot on one leg in the y-z and x-z planes (forwards and sideways tilt) when that foot is the only one on the ground. The main hip and hip side references are controlled, based on the current body tilts and x/y speeds. The $f_1$ and $f_2$ FOX modules adjusts these reference to fine tune the balance. Correct single-leg balancing tends to maintain or increase the current forwards/sideways speed, because the leg on the ground must be thrust outwards in the direction of travel to prevent the body angle from changing. This is demonstrated in figure 7.14 and movie `walk2.mpg`, which shows what happens when there is no hip side compensation.

**Direction changing module:** This tries to bring the biped's direction closer to the desired direction with each step. When the foot is placed on the ground and the other foot is lifted, the untwist_ref (table 7.4) angle is changed to bring the body around to the correct direction. When the foot is lifted again the reference is reset to zero. The $f_3$ FOX module adjusts the hip twist reference to stop the body from twisting when the foot is on the ground (see movie `walk3.mpg`).

**Leg placement controller:** This adjusts the main hip and hip side references when the foot is in the air, so that the leg position on foot impact will cause the long term balance to be maintained. The $f_4$ FOX module adjusts the main hip reference to ensure that this happens. Correct leg placement also tends to maintain the current speed. Figure 7.15 and movie `walk6.mpg` demonstrate the effect of leg placement.

**Toe collision avoidance module:** This prevents the foot from prematurely touching the ground during a step. The $f_7$ FOX module learns to cause a contraction of the leg (a hip-knee-ankle reference change) if the foot touches the ground during a part of the cycle when it is anticipated that the foot should be in the air (see movie `walk5.mpg`).

**Ankle support reaction:** This adjusts the ankle reference when there is a differential pressure between the heel and toe. This is done to help the biped keep its balance. For example, if the toe-pressure is greater than the heel-pressure the foot will be extended, increasing the toe-pressure in the short term but also hopefully pushing the body back to a more balanced position. This reaction also helps the biped to

push off with its rear foot in the moments before it takes a step with that foot. The set-point of this reflex can be adjusted to accelerate or decelerate the biped, by tending to tip it forwards or backwards. The $f_5$ FOX module adjusts this set-point to try and control the biped speed. The FOX's output is limited to prevent it from moving the set-point too far, which could cause over-balancing.

Another FOX module, $f_8$, adjusts the ankle reference to keep the biped upright. It is controlling the same parameter as $f_5$, so it needs to have a different eligibility profile to prevent the errors $e_1$ and $e_2$ from simply having an additive effect on the ankle's reference.

**Arm swing module:** This swings the arms to try and prevent the body from twisting during walking. The $f_6$ FOX module adjusts the gain of this reflex to get the best effect (see movie `walk3.mpg`).

**Global twist compensator:** This is common to both legs, and acts when both feet are on the ground to try and prevent body twist. It is essential, because the modules that control the main hip reference tend to "fight" each other when both feet are on the ground, causing a net torque to be applied between the feet which twists the body around. This twist can pull the feet out of their stable stance, causing the biped to slip. The $f_{G1}$ FOX module changes the main hip references of both legs (using a different sign for each leg) to try and counteract this twisting effect.

**Global side drift compensator:** This is also common to both legs, and acts when both feet are on the ground to tilt the body such that the sideways desired-path deviation is minimized. The appropriate hip side reference adjustment is learned by the $f_{G2}$ FOX module (see figure 7.16 and movie `walk4.mpg`).

The learning rates for all FOXs were chosen experimentally by observing the learned parameter. For example, if it did not grow fast enough, the learning rate was increased, and if it grew too large or started to over-train then the output limiting factor was increased.

### 7.7.4 Walking performance

As has already been seen, the biped can learn to walk with a steady gait. Figure 7.12 shows the training performance that was achieved with the system described here. A speed reference of 0.3 m/s and a constant direction reference were used. The time spent in each training iteration is shown (where each iteration ends when the biped falls over). After 220 iterations the biped is walking perfectly, with a total simulated training time of 74 minutes. The training time can be adjusted up and down by changing the learning rates. A compromise must be made: higher learning rates give faster learning, but the system is also more susceptible to learning abnormalities such as over-training and learning interference.

Notice that the biped's performance (iteration time) does not increase monotonically, as might be expected if this was a simpler optimal control system. Instead the biped goes through various training stages. In each stage a different failure mode is experienced (for example, falling to the side due to inadequate hip side compensation). Only a subset of learnable parameters experience a high training effect in each stage. Figure 7.17 shows a typical falling-over event near the start of training. A variety of such events are shown in movie `walk8.mpg`. Eventually the FOX modules have sufficient experience to keep the biped standing for extended periods of time, at which point the control parameters can be fine tuned to ensure a steady gait. Occasionally while the biped is walking it will get into an unanticipated configuration and fall over (figure 7.18), but these events are essential for the training experience to be comprehensive.

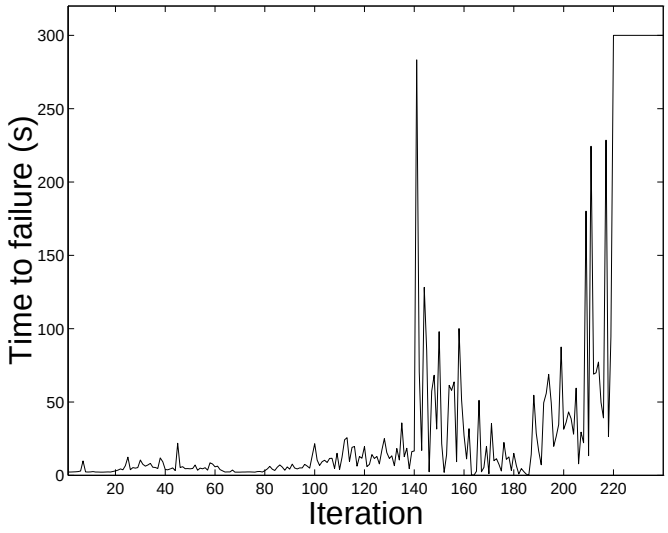


**Figure 7.12**: *The training performance for the walking biped. This shows the time spent in each training iteration, where each iteration ends when the biped falls over. After 220 iterations the biped walks perfectly (note the biped is reset after 300 seconds of successful walking). The total training time is 74 minutes.*

Once trained the robot was able to adapt its gait to speed references up to the reference used during training, as shown in movie `walkt2.mpg`. The biped was also able to follow an arbitrary path, simply by changing the direction reference and path deviation calculation (see figure 7.19 and movie `walkt2.mpg`).

The biped was also tested on its ability to walk up and down slopes. The test environment is shown in figure 7.20. After training the biped successfully crossed the ramp, as seen in figure 7.21, figure 7.22 and movie `walkt2.mpg`. The biped had to learn to adjust its gait to suit each slope separately. This is shown in figure 7.23, where the biped has successfully climbed the up-slope of the ramp but trips over on the flat top because it has not yet had flat-ground experience.

The effect that training has on the biped motion is subtle but significant. Figure 7.24 shows some biped joint angles over three stepping cycles, before training. The same thing after training is shown in figure 7.25. The general appearance of both plots is the same, that is the biped makes almost the same motions in each case. But on closer inspection there are many fine differences, which are required for stable walking.

Figure 7.26 shows three trained FOX outputs over three stepping cycles. The complex form of each FOX output has been acquired through training via the need to compensate as much as possible for the perturbing effects of the rest of the system (which includes the other FOX modules). In effect, the form of each signal is specially constructed to *anticipate* what the biped might be doing wrong and compensate for it before it has a chance to manifest itself. Also notice that the activity of each FOX output is concentrated in the region where its gating signal is nonzero.

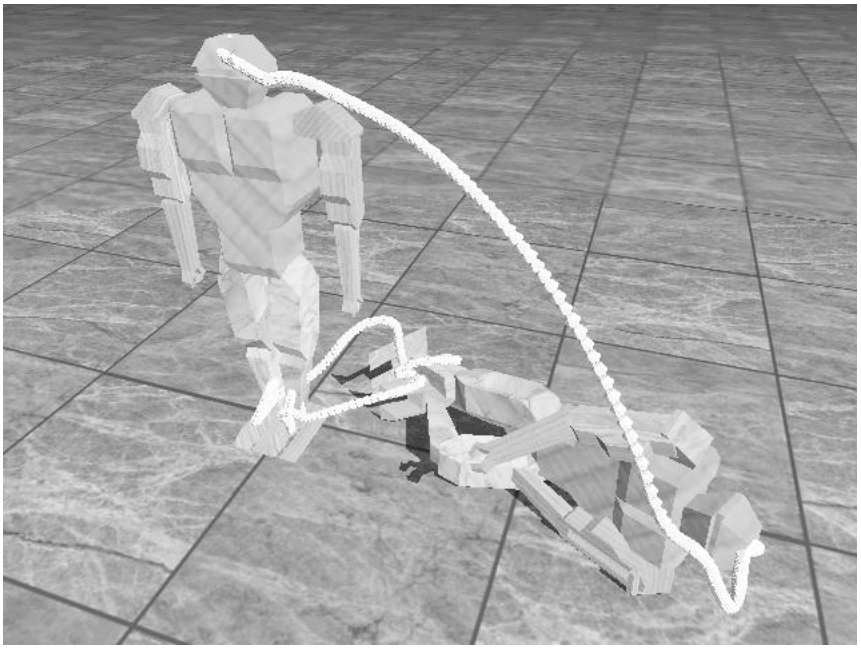

**Figure 7.13**: *Biped walking, where only the motion sequencer is used in the controller (no other controller modules are active). The biped makes stereotyped stepping movements and falls down immediately.*

**Figure 7.14**: *Biped walking, without hip side compensation. The biped sways from side to side as it walks (the trail left by the head has a large side-to-side variation).*

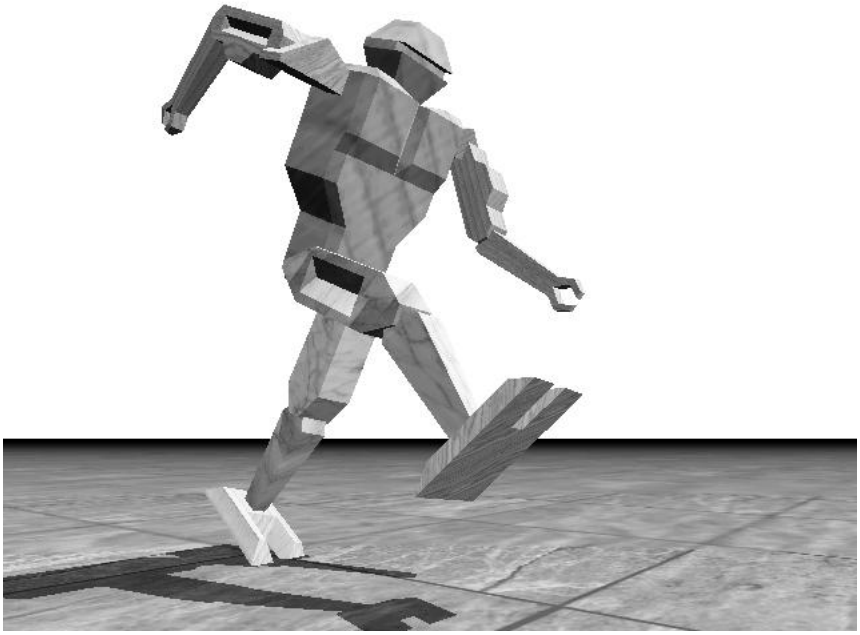

**Figure 7.15**: *Biped walking: this illustrates leg placement (the effect of $f_4$). When the foot is off the ground, the leg is placed to try and prevent a large body angle or forward speed.*

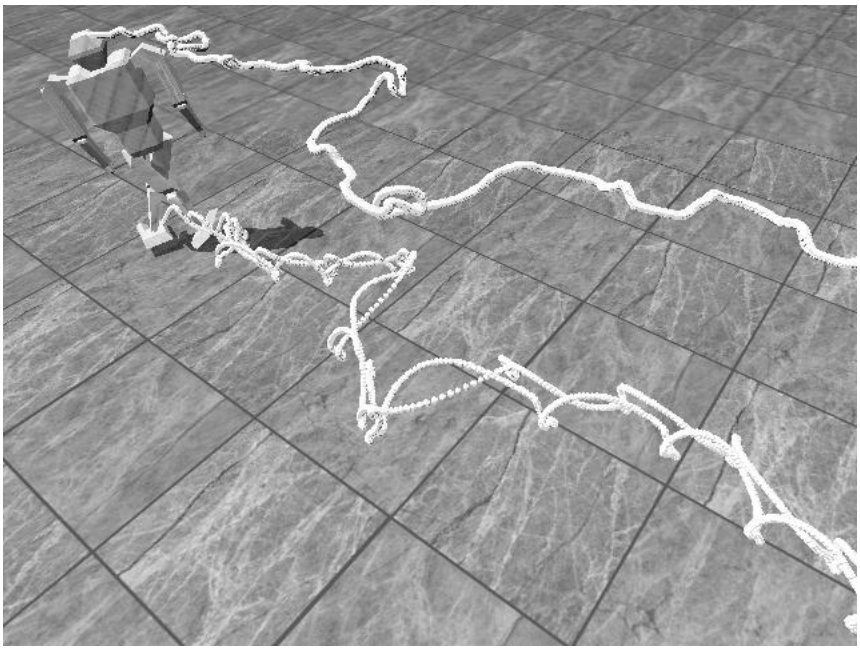

**Figure 7.16**: *Biped walking, without global side drift compensation. The biped strays from the desired path and becomes unstable trying to get back on it.*

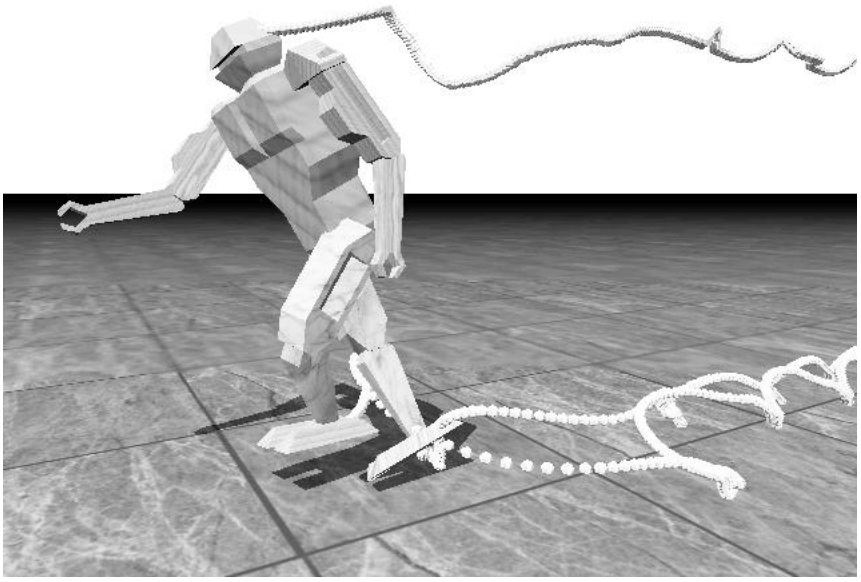

**Figure 7.17**: *Biped walking: this is a typical falling-over event near the start of training.*

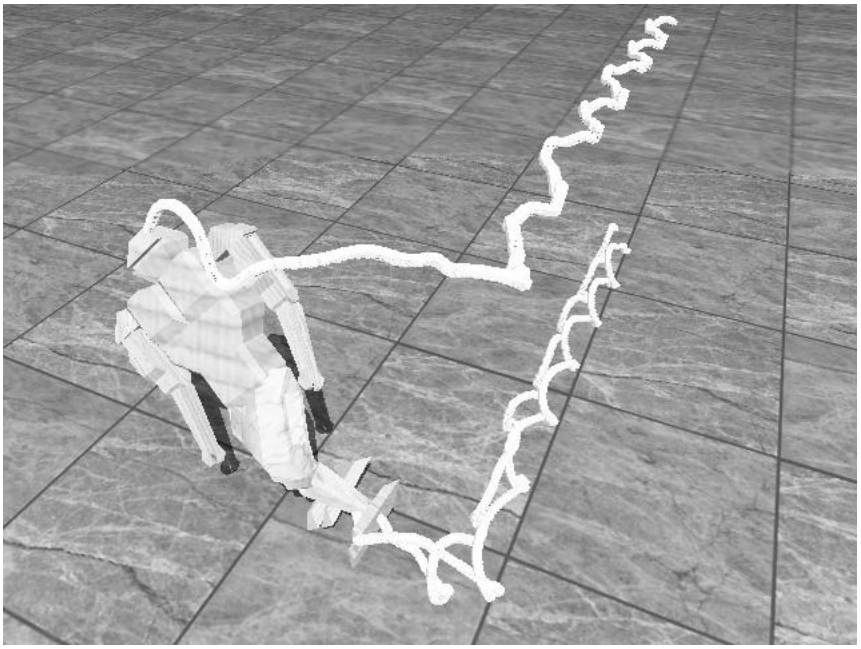

**Figure 7.18**: *Biped walking: this is a typical falling over event during training. The biped walks well for a while, then suddenly falls over.*

**Figure 7.19**: *Biped direction changing: the biped walks in a circle around the marker.*

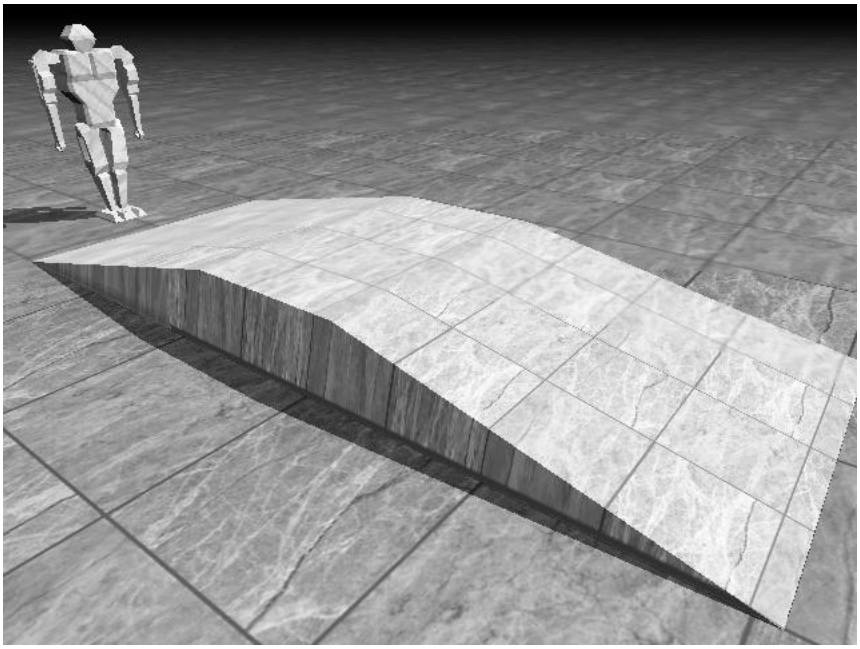

**Figure 7.20**: *Biped walking on slopes: this is the test environment, with a ramp the biped must climb and descend. The biped is in its starting state.*

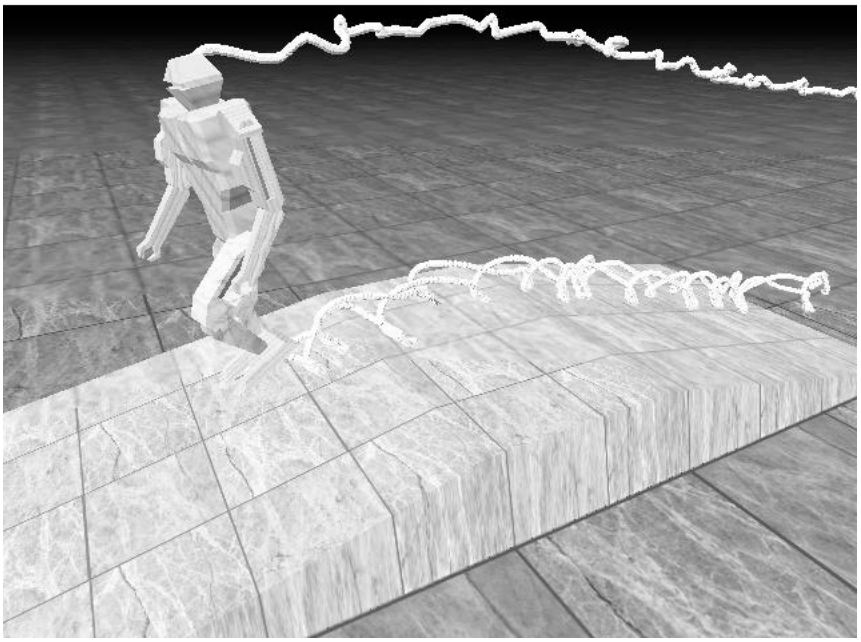

**Figure 7.21**: *Successful biped walking over the ramp.*

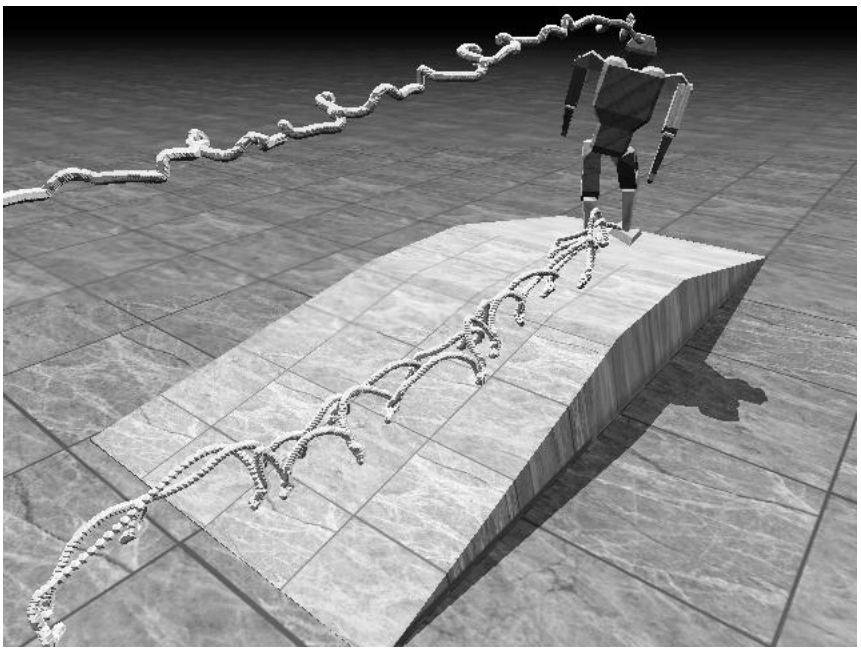

**Figure 7.22**: *Successful biped walking over the ramp, from another angle.*

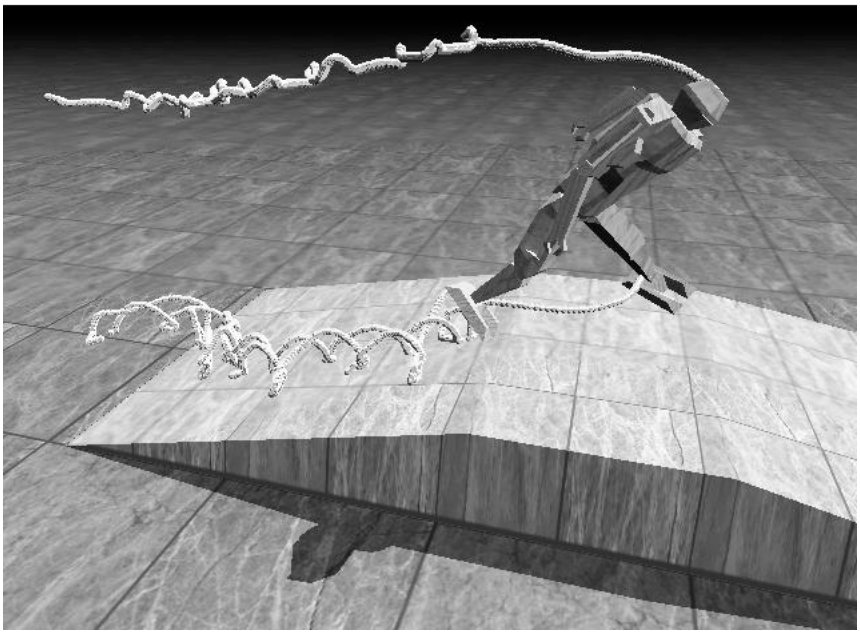

**Figure 7.23**: *Biped walking on slopes: during training the biped learns to walk on each gradient separately. Here it has successfully climbed the ramp but trips over at the top.*

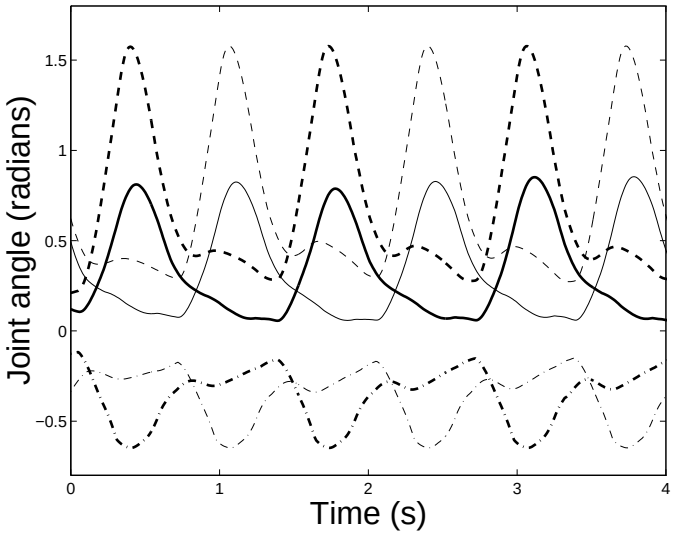


**Figure 7.24**: *Biped joint angles for three stepping cycles, before training. Key: left main hip joint (—), right main hip joint (**—**), left knee joint (– – –), right knee joint (**– – –**), left ankle joint (– · –), right ankle joint (**– · –**).*

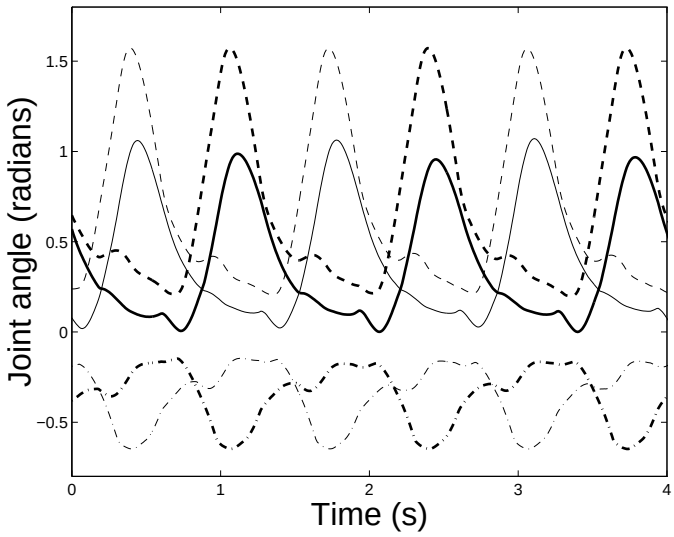


**Figure 7.25**: *Biped joint angles for three stepping cycles, after training. Key: left main hip joint (—), right main hip joint (▬), left knee joint (– – –), right knee joint (▬ ▬), left ankle joint (·····), right ankle joint (▬ · ▬).*

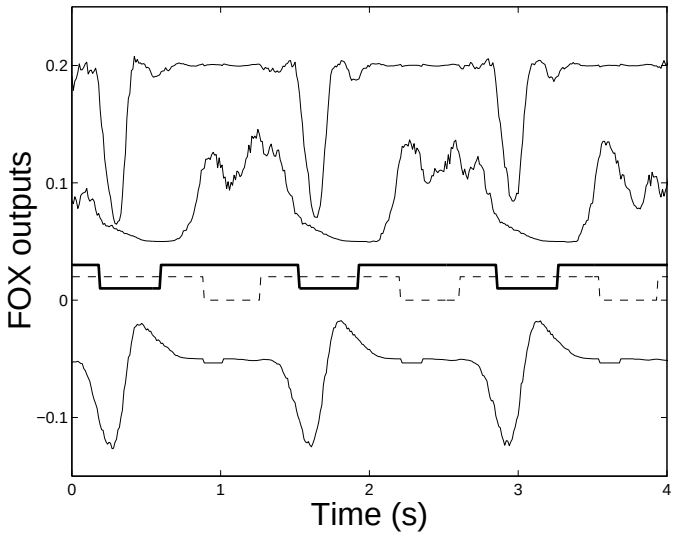


**Figure 7.26**: *FOX outputs for three stepping cycles, after training. Key: right leg FOX outputs (—), left foot sensor (– – –), right foot sensor (▬). The FOX outputs are, from top to bottom,* $f_2 + 0.2$, $f_4 + 0.05$, $f_1 - 0.05$.

## 7.8 Conclusion

The FOX controller can be used to successfully adjust control parameters to achieve stable walking in the biped robot. This is in spite of the fact that FOX is being used in a manner unanticipated by its theoretical formulation. There is no global error to be minimized, rather each FOX tries to improve some local error during part of the robot's motion. The eligibility profiles used by each FOX are derived from the designer's intuition (i.e. they are guessed), which is an acceptable procedure given the large amount of approximation that is allowed.

Once the biped was trained it had a similar performance to Laszlo's similarly configured biped [64], which used limit cycle control. But the system described here has the advantage that it can be trained on-line, and is thus far more suitable for implementation in a real robot.

The FOX error signals are not derived from a global error, so the biped's optimal behavior is not predefined. However, the biped's success can be judged in other ways: as it is trained it is able to walk for longer without falling over, it is able to walk more stably at the desired speed, and the body orientation deviates less from the desired orientation. Note that, as with the hopping robot, "perfect" control is never achieved because of the constraints imposed by the controller's internal structure.

The FOX modules are not given total control of the system, because the designer must constrain the robot's behavior via the controller design to whatever is appropriate for the environment. Because of this, perfect control is difficult to achieve, as not all the requirements can be satisfied simultaneously.

It is easy to assume that less care is needed when designing a FOX based controller, because FOX will be able to magically fix up any mistakes. But this is not really true: FOX modules make the system more adaptive, but care and attention is still needed to design a practical controller.