NASA-CR-199236

Lamar University

BEAUMONT, TEXAS

TECHNICAL REPORT

(NASA-CR-199236) LOCOMOTION TRAINING OF LEGGED ROBOTS USING HYBRID MACHINE LEARNING TECHNIQUES Final Report (Lamar Univ.) 84 p

N96-10100

Unclas

G3/63 0064233

COLLEGE OF ENGINEERING

Locomotion Training of Legged Robots Using Hybrid Machine Learning Techniques

FINAL REPORT
NASA Grant NAG 9-695

September 1, 1995

Lamar University
College of Engineering
Beaumont, Texas

William E. Simon, Ph.D., P.E., Mechanical Engineering, Principal Investigator
Peggy I. Doerschuk, Ph.D., Computer Science, Research Associate
Wen-Ran Zhang, Ph.D., Computer Science, Research Associate
Andrew L. Li, Graduate Research Assistant

Abstract

In this study artificial neural networks and fuzzy logic are used to control the jumping behavior of a three-link uniped robot. The biped locomotion control problem is an increment of the uniped locomotion control. Study of legged locomotion dynamics indicates that a hierarchical controller is required to control the behavior of a legged robot. A structured control strategy is suggested which includes navigator, motion planner, biped coordinator and uniped controllers. A three-link uniped robot simulation is developed to be used as the plant. Neurocontrollers were trained both online and offline. In the case of on-line training, a reinforcement learning technique was used to train the neurocontroller to make the robot jump to a specified height. After several hundred iterations of training, the plant output achieved an accuracy of 7.4%. However, when jump distance and body angular momentum were also included in the control objectives, training time became impractically long. In the case of off-line training, a three-layered backpropagation (BP) network was first used with three inputs, three outputs and 15 to 40 hidden nodes. Pre-generated data were presented to the network with a learning rate as low as 0.003 in order to reach convergence. The low learning rate required for convergence resulted in a very slow training process which took weeks to learn 460 examples. After training, performance of the neurocontroller was rather poor. Consequently, the BP network was replaced by a Cerebellar Model Articulation Controller (CMAC) network. Subsequent experiments described in this document show that the CMAC network is more suitable to the solution of uniped locomotion control problems in terms of both learning efficiency and performance.

A new approach is introduced in this report, viz., a self-organizing multiagent cerebellar model for fuzzy-neural control of uniped locomotion is suggested to improve training efficiency. This is currently being evaluated for a possible patent by NASA, Johnson Space Center.

An alternative modular approach is also developed which uses separate controllers for each stage of the running sride. A self-organizing fuzzy-neural conroller controls the height, distance and angular momentum of the stride. A CMAC-based controller controls the movement of the leg from the time the foot leaves the ground to the time of landing. Because the leg joints are controlled at each time step during flight, movement is smooth and obstacles can be avoided. Initial results indicate that this approach can yield fast, accurate results.

Table of Contents

Page

List of Figures

List of Tables

SECTION 1

Introduction

Legged locomotion is superior to wheeled locomotion in terms of versatility and flexibility. The development of legged robots having the same locomotion capabilities as humans remains an outstanding challenge for researchers in robotics. Several research teams have focused on the development of legged robots in the U.S., Japan, and Europe. Since the 1970's several legged robots have been built with the ability to walk, run, and even do gymnastics, while the major control strategies for legged robots have been classical control techniques. Now, however, exploration of machine learning techniques applied to legged locomotion is beginning to show promise.

The two primary modes of legged locomotion, walking and running, are distinguished by their nature. Walking is characterized by the constant contact of at least one foot with the ground, with requisite periods of multifoot contact. Biped walking motion can be considered as repetitions of two phases: a single-support phase followed by a double-support, or changeover, phase. Running, on the other hand, is characterized by the contact of at most one foot with the ground at any time. Running also involves two phases: a ballistic phase in which there is no foot in contact with the ground, and a support phase in which exactly one foot is in contact with the ground. When biped running is reduced to one leg, uniped jumping results. Conversely, biped running can be viewed as the coordination of alternating uniped jumps. The study of running can therefore be simplified to uniped jumping without altering the nature of the problem.

To date, most control strategies used for legged locomotion have been based on linear, nonadaptive control equations or rules of motion. These kinds of controllers lack robustness with respect to environmental variations and disturbances. In the last few years, computational intelligence has been introduced into the robotic controls area, and this has yielded encouraging results. One particular form of computational intelligence, neural networks, has been widely used due to its features of parallel processing, nonlinear mapping, learning ability, and generalization. A neural network is composed of processors, called neurons, connected with weighted synapses. Neurons are usually arranged in layers.

Neurocontrollers are controllers based on neural networks and can be classified into two groups with respect to their learning algorithms: online and offline. In the online approach, the neurocontroller is trained alongside the plant during normal operation. With offline training, the training data is collected during plant operation and used to train the neural net regardless of the operational state of the plant. Both approaches have been applied to robotic controls with various degrees of success. More recently, a new class of neural networks, called the Cerebellar Model Articulation Controller (CMAC), has proved superior to traditional neural networks in control applications. CMAC was developed from models of human memory and neuromuscular control. Unlike traditional neural networks, a CMAC network contains no neurons. Its capabilities for the adaptation of nonlinear modeling and generalization are embedded in the manner in which the CMAC maps points in the input space into an associative memory. Nearby points are mapped into nearby positions with partial overlap, while points far away from one another are mapped into remote regions of the associative memory. According to one study (Brown and Harris, 1994), CMAC actually lies between the traditional neural net and the fuzzy rule base.

In recent years, several research teams have successfully applied neural networks to robot arm manipulation, but very rarely have attempts been made to apply it to legged locomotion. In September 1993, Lamar University was awarded a grant from the Johnson Space Center (JSC) to explore the application of intelligent machine-learning techniques to the uniped locomotion problem. One of the objectives of this project was to develop uniped control strategies based on neural networks. The achievements of this study include:

1) building a three-link uniped robot simulation and using the simulation to analyze the dynamics of uniped locomotion;
2) evaluating the control signal requirements and designing a hierarchical controller;
3) constructing and training various neural networks as modular components of this hierarchical controller;
4) comparing these neural networks in terms of training efficiency and performance; and
5) and providing insight for future studies of uniped and biped locomotion.

SECTION 2

Literature Review

Legged locomotion requires the exertion of a traction force in a direction opposite to that of motion. The reactive force thereby provides propulsion for forward motion (Todd, 1985; Raibert, 1986). This is usually accomplished by applying opening torques at the joints. Figure 2.1 shows that, when running, the exerted torques at the knee and ankle tend to enlarge these angles and result in motion of the foot opposite to the direction of the body motion. The reactive force caused by the motion of the foot is equal in magnitude to the force exerted by the foot on the surface. One of the main characteristics of legged locomotion is that joint torques compose the main control parameters.

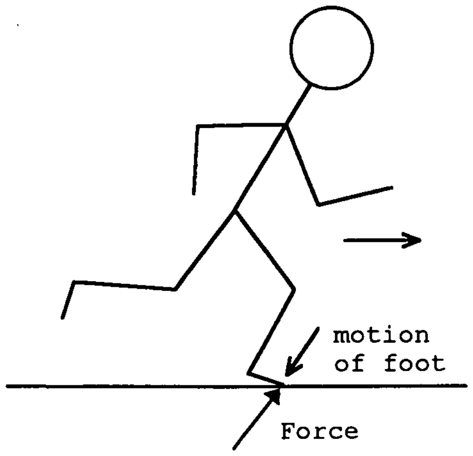

Figure 2.1 Reactive Force

Research in the area of legged locomotion began many years ago. To date, significant results have been generated. So far, much effort has been expended developing multiped robots for use in extreme environments. On the other hand, theoretical studies in biped locomotion have resulted in numerous publications on many aspects of this problem. All in all, very few biped robots have actually been built, due to their complexity in both theory and practice. In this chapter, some research work in biped locomotion published recent years is reviewed. In addition, the achievements of research in applications of artificial neural networks to robot manipulator control are also summarized.

2.1 Biped Locomotion and Control Strategies

Since the 1970's, biped legged robots have been built with various locomotion abilities in the U.S., Japan, and Europe. Research in legged locomotion can be categorized as walking versus running in terms of motion type, and also as adaptive versus nonadaptive in terms of control strategies. So far, the majority of the research groups have been focusing on nonadaptive walking control.

The work of Zheng is typical of this approach (Zheng, 1989, 1990; Goddard and Zheng, 1992). After developing one of the earliest biped robots, called CURBi, Zheng in 1987 built a follow-on robot called SD-2, which is a 5-link biped with eight degrees of freedom. He also developed a feedback control system to allow the robot to walk on level ground as well as well as on sloping surfaces. This was an attempt leading to the solution of the difficult control problem of walking on irregular terrain. After solving the external disturbance rejection problem, Zheng successfully made the robot walk from a level to a sloping surface. He used a nonadaptive control strategy composed of three parts. The first part was a scheme for detecting and measuring the gradient of the slope. The second part dealt with the walking gait on the slope, and the third and most significant aspect controlled the robot as it walked through the transition area joining the level and the sloping surfaces. Zheng's control algorithms were based on the equations of motion of the robot and the signals from sensors mounted on the robot's feet.

Several research groups in Japan have also been working on biped robots. Furusho built a 9-link biped and its control system (Furusho and Sano, 1990). First, he used the so-called "Reduced Order Model" to simplify and examine the walking system. The anthropomorphic 9-link biped robot he developed weighs 25 kg and stands at 0.97 m, with eight degrees of freedom. Furusho chose a sensor-based feedback control method with signals from foot pressure sensors, ankle torque sensors, inclinometers, angular rate sensors, speed sensors, and accelerometers. In this control strategy, the motion of the robot was divided into two planes: the sagittal plane (vertical to the floor in the direction of walking) and the lateral plane (perpendicular to the direction of walking). This division reduced the complexity of the control problem through decomposition of motion in two orthogonal planes. In the lateral plane, the motion was simply a repetitive tilting to alternately place the center of mass directly over the left or right supporting foot. In the sagittal plane, the main control objective of walking motion is the body speed in forward motion. Unlike Zheng's SD-2 robot, Furusho's robot could only walk on flat surfaces.

Shih presented another control strategy for a 7-link biped which had twelve degrees of freedom (Shih and Gruver, 1992). Shih noticed that most biped control strategy research was concentrated on the single-support phase, while in biped walking, the double and single-support phases alternate. Shih hypothesized that the analysis of biped locomotion in the double-support phase is important for improving the smoothness of the biped locomotion system. Shih implemented this idea by introducing a reduced-order model with constraints, in which the selected dependent variables were related to independent variables through the kinematic Jacobian. A control strategy based on feedforward compensation and linear state feedback was used to track the desired trajectory.

For several years, Raibert and his colleagues at the MIT Artificial Intelligence Laboratory have been the sole individuals working on the problem of biped running. Their running robots, some of which can even do gymnastics, rely on telescoping legs and simple nonadaptive, staged-control strategies (Hodgins, 1990). This type of robot is controlled by a feedforward controller which works well in a disturbance-free environment. The feedforward control signals are generated analytically or empirically. The analytical approach is based on the following steps:

1. modeling of the running machine;
2. finding the appropriate pitch rate using the equations;
3. calculating the required forward speed and the pitch rate of the body required; and
4. calculating the appropriate torques and output signals.

The empirical approach uses examples for learning. However, this work was done essentially by humans based on knowledge of mechanics and on intuition, instead of by the robot via machine learning techniques.

Work on adaptive walking control was led by Wang, who developed a neurocontroller for a 3-link biped robot (Wang and Malakooti, 1993). Wang used the equations of motion and control rules to train a group of traditional backpropagation (BP) neural networks rather than to control the robot directly. Two different architectures were investigated, and their properties analyzed. This work is a significant beginning, even though the robot is not as sophisticated as some with conventional control strategies.

Miller published his research on the biped gait controller (Miller, 1994), and he was the first to apply CMAC networks in biped locomotion control. CMAC networks were trained in several aspects of locomotion control including: (1) closed-chain kinematics necessary to shift body weight side-to-side while maintaining adequate foot contact; (2) quasi-static balance to avoid falling forward or backward while shifting body weight side-to-side at different speeds; and (3) dynamic balance required to lift the foot off the floor for a desired length of time, during which the foot can be moved to a new location relative to the body. This is an important contribution to biped locomotion research, and these experiments confirmed the applicability of computational intelligence in legged locomotion control.

2.2 <u>Neural Networks in Robotic Control</u>

The last twenty years saw phenomenal growth of artificial neural network applications in many different areas. Neural networks attempt to simulate in some aspects the working of biological brains, whose characteristics include parallel processing, learning ability, nonlinear mapping, and generalization. In the field of robotic control, many attempts have been made to utilize neural networks. Fukuta reviewed the development of neural networks and their application in industrial control systems (Fukuta and Shibata, 1992). The application of neural networks to control problems can be categorized into a number of major distinct groups: supervised control, inverse control, neural adaptive control, backpropagation of utility (an extension of backpropagation through time), and adaptive critics (an extension of reinforcement learning). In fact, many applications involve combinations of one or more of the methods listed above.

In supervised control, a neural network learns the mapping from input signals to desired control actions by adapting to a training set. In inverse control, the neural network learns the inverse dynamics of a plant without any symbolic description of the system. In neural adaptive control, a neural network is used in a place of more classical mappings. Backpropagation of utility and adaptive critics are two general-purpose designs for optimal control using neural networks. In both backpropagation of utility and adaptive critics, the user specifies a utility function or performance index to be maximized, or a cost function to be minimized. Out of these methods, supervised control, inverse control, backpropagation, or hybrid combinations of these are most frequently used in robotic control. Almost all neural networks, except the CMAC networks, which have been applied to robotic control are feedforward multilayered networks.

A neurocontroller for autonomous underwater vehicle control was built with a combination of the methods of inverse dynamic learning and backpropagation (Venugopal, Sudhakar and Pandya, 1992; Venugopal, Pandya and Sudhakar, 1994). The objective of this controller was to make the vehicle track, with minimum error, a desired trajectory related to corresponding displacements of the angular velocities along three axes. The neurocontroller was given a desired command signal concerning the trajectory, after which it "learned" the inverse dynamics, without a definition, of the vehicle to produce a correct control input. It is reported that with this control strategy, the vehicle follows the desired trajectory very precisely. In addition, Venugopal and his coworkers presented an application of a learning algorithm called Alopex in which weight updating is based on the system output error directly, rather than using a transformed version of the error.

Another type of vehicle, the mobile robot, was also united with neural net control in a Fujitsu laboratory in Japan (Nagata, Sekiguchi and Asakawa, 1990). This four-wheeled robot was controlled by a special multilayered neural network structure based on signals from twelve sensors mounted on the robot. The network model was divided into two subnetworks, named Reason Network and Instinct Network, connected to each other by short-term memory units. This structure divides one complex task into two, and with several mutual constraints it enables the networks developed for different tasks to cooperate efficiently.

Other applications of neural network controls have been in the area of robotic manipulators. An early and typical example of the inverse control method was given by Miyamoto (Miyamoto, Kawato, Setoyama and Suzuki, 1988). A neural network was used to learn the inverse dynamics of a robot manipulator through backpropagation. After the network was trained, it generated correct control signals to the manipulator to perform specified actions. The learning rate was reported as follows: "The performance by the neural network model improved gradually during 30 minutes of learning." The reason that the Miyamoto neural net is able to learn the inverse dynamics of this manipulator in such a short time is that the dynamic relationship between torques and behavior of the two-link robot arm, both forward and inverse, are relatively linear. In a general case involving more severe nonlinearities, the training rate is expected to be significantly slower.

Computational intelligence techniques have also been used cooperatively with traditional AI expert systems in practice. A control algorithm which combined a knowledge-based AI system and a CMAC network for the control of a two-link robotic manipulator was published (Handelman, Lane and Gelfand, 1990). The author realized that although neural networks have proven to be very efficient in the learning process, the training data, in general, must be supplied by an outside operator who must also closely supervise the learning process. The basic idea is that, while the neural net has sufficiently been trained with examples generated by the expert system, the neural network's ability to generalize is what makes it a good controller. In essence, the learning process of this neurocontroller was supervised by a knowledge base, rather than using inverse dynamic learning.

Another control algorithm for a robot manipulator used a CMAC network as a compensator in a control system (Miller, Glanz and Kraft, 1987; Miller, 1990). The objective of this control system was to have a two-link robot track desired trajectories, with the torques at the two joints as control parameters. In this algorithm, a control signal is generated by a fixed-gain feedforward controller. This signal is combined with a compensation signal produced by a CMAC network. This online learning algorithm was found to provide good dynamic performance in complex situations.

Another example of the use of neural networks as compensators is found in a different form (Ishiguro, Furuhashim, Okuma and Uchikawa, 1992). Based on the realization that generating examples for neural network training may be different when the behavior of the plant is not known and on the belief that neural networks perform best when they are not required to learn very much, Ishiguro and his colleagues proposed a new control strategy for a robotic manipulator. In this strategy, neural networks are incorporated as compensators and are required to learn not the inverse dynamics, but instead the structured and unstructured uncertainties of the manipulator. These uncertainties are the principal factors which degrade high-speed performance. Since the approximate model of the manipulator can be derived, and model-based control can be an effective approach to high-performance control, these researchers chose to use a model-based control method with neural networks trained to compensate for uncertainties. The "computed torque method" and a method to obtain the "true teaching signals" were outcomes of this research. The neural networks must only learn the compensations to the computed torques. Learning, therefore, became simpler. However, this algorithm was limited to those plants whose dynamic models are relatively simple.

Reinforcement learning is another important controller training technique. Neurocontrollers can be trained with supervised methods only when the correct control action examples are available. Unfortunately, in many situations it is difficult to obtain these examples. On the other hand, an index of performance can often be defined in a straightforward manner and can be used to drive the reinforcement learning process (Morgan, Patterson and Klopf, 1990; Wu and Pugh, 1993). Examples of reinforcement learning algorithms were presented by Gullapalli et al, 1994. Two neural networks were trained to perform two tasks: peg-in-hole insertion and ball balancing. The results showed that the trained neurocontrollers successfully performed these tasks. The key to reinforcement learning is to specify the performance evaluation function, by which the "error" of the neural net is computed to provide guidance for weight updating.

Publications in legged locomotion and applications of neural networks to robotic control are growing every day. Among these publications, only a representative sample was selected as reference for this study. In legged locomotion research, traditional approaches use the process of inverse dynamics, which calculates the control parameters required to perform a specified motion. Inverse dynamics gives numerical, as opposed to analytical, results (Roberson, 1988; Asada, 1990). In general, a great deal of computation is involved. When environmental conditions are unknown or variable, this approach becomes unsuitable. Neural networks avoid the computations of inverse dynamics, but instead learn from experience just as a human does. Although neural networks have been used extensively in robotic manipulator controls, their use in the control of legged locomotion has only recently begun. Reasons for this slow development include greater complexity of the control problems and the large amount of information to be processed.

SECTION 3

Uniped Dynamics

A software package was developed to simulate the dynamics of a three-link uniped. Several issues concerning the dynamics of this model are discussed in this chapter.

3.1 Why Two-Dimensional, Uniped, and Three-Link?

The purpose of this study is to investigate control strategies for legged locomotion. As such, it is proper to begin with as simple a model as possible without altering the essence of the locomotion problem. A model as complicated as a three-dimensional (3D) human-like robot is not suitable for this study in its first stage. Instead, a two-dimensional (2D) model seems proper, since 3D locomotion can be synthesized from a 2D model combined with a balance algorithm in the third coordinate.

It has been reasoned in the first chapter that running is basically a coordinated jumping problem. This means that the biped model is an increment of the uniped model. A uniped control problem includes the primary difficulties of the biped control problem if running is the locomotion mode of interest.

Concerning the robot structure, a two-link mechanism is too simple to generate legged locomotion. A four-link model is more human-like, but not as suitable as the three-link model to begin with because it involves more control parameters and state variables than are necessary. Experiments with a three-link robot indicate that such a robot displays sufficient complexity worthy of consideration. The schematic of the robot in Figure 3.1 shows that instead of having a human-like configuration, the three-link uniped looks rather like the rear leg of a cat.

3.2 The Structure of the Robot in Simulation

As shown in Figure 3.1, the robot's body mass, represented by the large circle in the figure, is lumped at the extremity of the top link. Table 3.1 lists the kinematic and dynamic properties of the robot. For the robot to jump, joint torques (at ankle and knee) are required. Nonlinear torsion springs are used to model joint torques as well as joint angle restrictions.

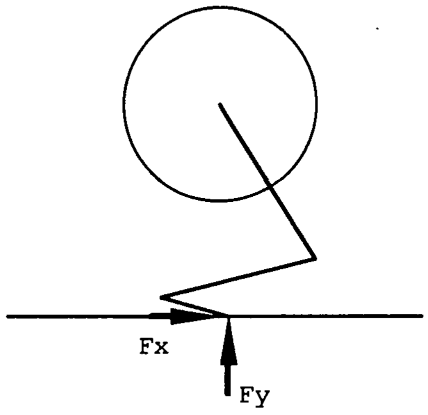

Figure 3.1 Three-Link Uniped Robot

Table 3.1 Kinematic and Dynamic Properties of the Robot

Link	L (m)	l (m)	m (Mass) (kg)	Moment of Inertia (kg-m2)	Initial Orientation(deg)
Foot	0.2	0.1	0.5	0.005	180
Lower Leg	0.4	0.2	1.0	0.040	20
Thigh(Body)	0.4	0.4	6.0	0.300	variable

The physical meanings of L and l are shown in Figure 3.2.

L -- length of a link;

l -- dimension from the center of mass to the end pointed;

Ground impact and support are modeled with nonlinear extensional springs which supply normal reaction forces and friction forces at the time of surface contact. Surface compliance variability is accounted for by changing the spring constants.

3.3 Equations of Motion

The three-link uniped robot and its motion can be described with a set of differential equations. Figure 3.2 shows the free-body diagrams for the robot's links, and dimensional parameters. The values of L and l for each link are listed in Table 3.1. Those parameters not shown in Figure 3.2 will be explained as they arise in the development of the equations.

Among the forces, *PX1, PY1, PX2* and *PY2* are reactive forces from the ground acting on the foot, and *FX1, FY1, FX2* and *FY2* represent internal reaction forces at the joints. *M1* and *M2* are the torques applied to the joints to implement the jump and to swing the links during flight. Position and dimension measurements are represented by $\theta 1$, $\theta 2$ and $\theta 3$ (orientations of the links) and *O1, O2* and *O3* (positions of the mass centers for the three links).

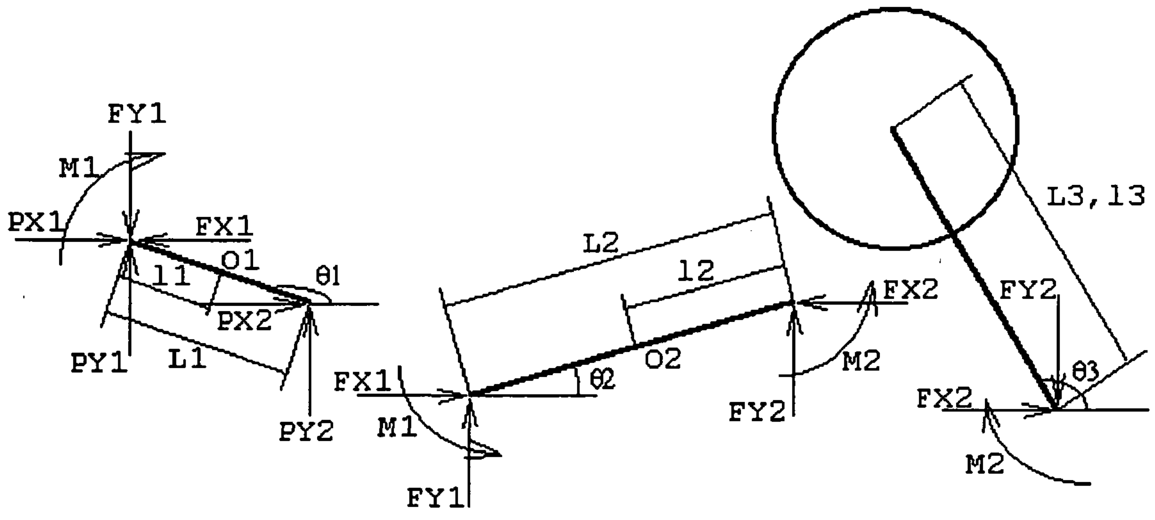

Figure 3.2 Forces and Dimensions of Each Link

From Newton's second law, there arise the following equations:

$$a_x 1 = (PX1 + PX2 - FX1)/m \tag{3.1}$$

$$a_x 2 = (FX1 - FX2)/m2 \tag{3.2}$$

$$a_x 3 = FX2/m3 \tag{3.3}$$

$$a_y 1 = (PY1 + PY2 - FY1)/m \tag{3.4}$$

$$a_y 2 = (FY1 + FY2)/m2 \tag{3.5}$$

$$a_y 3 = -FY2/m3 \tag{3.6}$$

$$\alpha 1 = \frac{\begin{array}{c}(-M1 + ((FX1 - PX1)Sin\theta 1 + (FY1 - PY1)Cos\theta 1)l1 \\ + (PX2 \cdot Sin\theta 1 + PY2 \cdot Cos\theta 1)(L1 - l1))\end{array}}{I2} \tag{3.7}$$

$$\alpha 2 = \frac{\begin{array}{c}(M1 + M2 + (FX1 \cdot Sin\theta 2 - FY1 \cdot Cos\theta 2)(L2 - l2) \\ + (FX2 \cdot Sin\theta 2 + FY2 \cdot Cos\theta 2)l2)\end{array}}{I2} \tag{3.8}$$

$$\alpha 3 = \frac{-M2 + (FX2 \cdot Sin\theta 3 - FY2 \cdot Cos\theta 3)l3}{I3} \tag{3.9}$$

Relations between acceleration, velocity, and position are included in the following equation set:

$$a_x 1 = dv_x 1 / dt \tag{3.10}$$

$$a_x 2 = dv_x 2 / dt \tag{3.11}$$

$$a_x 3 = dv_x 3 / dt \tag{3.12}$$

$$a_y 1 = dv_y 1 / dt \tag{3.13}$$

$$a_y 2 = dv_y 2 / dt \tag{3.14}$$

$$a_y 3 = dv_y 3 / dt \tag{3.15}$$

$$v_x 1 = dx1 / dt \tag{3.16}$$

$$v_x 2 = dx2 / dt \tag{3.17}$$

$$v_x 3 = dx3 / dt \tag{3.18}$$

$$v_y 1 = dy1 / dt \tag{3.19}$$

$$v_y 2 = dy2 / dt \tag{3.20}$$

$$v_y 3 = dy3 / dt \tag{3.21}$$

$$\alpha 1 = d\omega 1 / dt \tag{3.22}$$

$$\alpha 2 = d\omega 2 / dt \tag{3.23}$$

$$\alpha 3 = d\omega 3 / dt \tag{3.24}$$

$$\omega 1 = d\theta 1 / dt \tag{3.25}$$

$$\omega 2 = d\theta 2 / dt \tag{3.26}$$

$$\omega 3 = d\theta 3 / dt \tag{3.27}$$

Conditions of constraint at the joints are:

$$x1 + l1 \cdot Cos\theta 1 = x2 - (L2 - l2)Cos\theta 2 \tag{3.28}$$

$$y1 + l1 \cdot Sin\theta 1 = y2 - (L2 - l2)Sin\theta 2 \tag{3.29}$$

$$x2 + l2 \cdot Cos\theta 2 = x3 - l3 \cdot Cos\theta 3 \tag{3.30}$$

$$y2 + l2 \cdot Sin\theta 2 = y3 - l3 \cdot Sin\theta 3 \tag{3.31}$$

Variables related to reaction forces from the ground to the foot but not illustrated in Figure 3.2 are defined as:

$$X_{toe} = X1 - (L1 - l1)Cos\theta 1$$

$$Y_{toe} = Y1 - (L1 - l1)Sin\theta$$

$$X_{heel} = X1 + l1 \cdot Cos\theta$$

$$Y_{heel} = Y1 + l1 \cdot Sin\theta 1$$

$$y_{toe} = V_y 1 - (L1 - l1) \cdot \omega 1 \cdot Cos\theta$$

$$y_{heel} = V_y 1 + l1 \cdot \omega 1 \cdot Cos\theta$$

The reactive forces of the ground are:

$$\begin{aligned} PY1 = 0; \quad & \quad if\ Y_{heel} \geq 0 \\ K(-Y_{heel}); \quad & \quad if\ Y_{heel} < 0,\ \&\ Vy_{heel} > 0 \\ K(-Y_{heel}) + \mu Vy_{heel}\ ; \quad & \quad if\ Y_{heel} < 0,\ \&\ Vy_{heel} \leq 0 \end{aligned} \tag{3.32}$$

$$PY2 = \begin{cases} 0; & \text{if } Y_{toe} \geq 0 \\ K(-Y_{toe}); & \text{if } Y_{toe} < 0, \ \& \ Vy_{toe} > 0 \\ K(-Y_{toe}) + \mu Vy_{toe}; & \text{if } Y_{toe} < 0, \ \& \ Vy_{toe} \leq 0 \end{cases} \tag{3.33}$$

$$PX1 = \begin{cases} 0; & \text{if } Y_{heel} \geq 0 \\ K(X_{heel} - X0_{heel}) + \mu Vx_{heel}; & \text{if } Y_{heel} < 0 \end{cases} \tag{3.34}$$

$$PX1 = \begin{cases} 0; & \text{if } Y_{toe} \geq 0 \\ K(X_{toe} - X0_{toe}) + \mu Vx_{toe}; & \text{if } Y_{toe} < 0 \end{cases} \tag{3.35}$$

where:

$$X0_{heel} = X_{heel}$$

at the instant when the *heel* touches the ground, and

$$X0_{toe} = X_{toe}$$

at the instant when the *toe* touches the ground. The surface condition is described by constants *K* and μ, which can be varied to reflect different ground conditions.

3.4 Isolation of a Single Step Jump

The condition for continuous jumping is that at the end of a current jump, a proper landing configuration in terms of kinetics and position must be achieved in order to begin the next jump. This configuration includes linear velocity, orientation, and angular velocity for each link. To begin, an isolated jump is considered as a mathematical function. The inputs to this function are the initial configuration and joint torques. The outputs are the jump height, jump distance, and landing configuration. The term "initial configuration" also includes velocity, orientation, and angular velocity for each link.

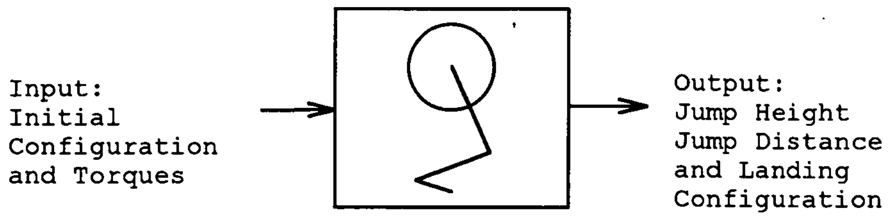

Figure 3.3 Robot Function: Input and Output

Jump height is measured as the vertical position of the center of mass when the robot's center of mass reaches its highest point in the trajectory. Jump distance is defined as the advance of the robot's mass center between take-off and landing. The landing angle, defined as the angle between the straight line connecting the foot center to the mass center and the vertical, must fall within a specified envelope (Figure 3.4). The angular momentum of the robot during flight is also controlled to prevent excessive rotation, which could lead to somersaults; and also to make it possible for the landing angle to fall within the specified envelope. The control parameters consist of joint torques and the initial configuration of the robot.

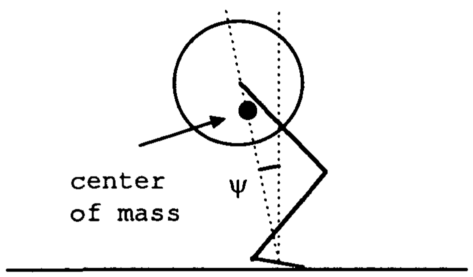

Figure 3.4 Landing Body Angle

3.5 Decomposition of a Single Jump

A single jump can be thought of as occurring in two phases. During the support phase, the foot is in contact with the ground and the joint torques are exerted to move the body into a crouch position for the jump. Large joint torques are then exerted to initiate the ballistic phase. During the ballistic phase, there is no interaction between the robot and the surrounding environment (air resistance is ignored); thus, due to the law of conservation of momentum, there is nothing the robot can do to change its overall motion, such as jump height and angular momentum (This will be discussed further in section 3.7.). As an analogy, when a baseball is thrown, not only its flight height and angular momentum, but also its flight distance is determined at the moment it begins the flight process. However, the problem of the jumping robot is not identical to the flight of the baseball because a robot has a changing radius from the center of mass to the point at which it contacts the ground, due to the changing of its link configuration. Therefore, different swing motions of the links may result in a slight difference in jump distance. The purpose of the swinging motion during the ballistic phase is to maneuver its links into place for landing. This can be accomplished with a single set of joint torques at the apex of the trajectory if the objective of ballistic control is simply the proper landing position. Otherwise, it can be accomplished with a sequence of joint torques if the objective is a specified sequence of positions along the trajectory for the purpose, for example, of avoiding obstacles.

3.6 Simplification of the Problem

In order to start with as simple a model as possible, the initial configuration of a single jump can be such that the initial velocity and angular velocity of each link are zero. A further simplification is made by fixing the initial orientation of the two lower links (see Table 3.1). Thus, the number of control parameters is significantly reduced (Once this simplified case is completely solved, the continuous jump can be addressed by releasing constrained parameters included in the initial configuration of the single jump.). The thigh orientation is retained as a control parameter to allow the robot to control the initial placement of its center of mass with respect to is foot. The other control parameters are the initial joint torques, and torques during flight. Since the robot cannot change its linear and angular momentem values during the ballistic phase, its gross motion is basically determined by initial torques and initial thigh orientation.

Initially, a simple PD (Proportional and Derivative) controller is used to control the body configuration during the ballistic phase. The PD controller responds to desired joint angles and generates a sequence of joint torques to achieve these angles. It is assumed that the PD controller can reach the objective with satisfactory precision (Kuo, 1991). The PD controller can be described as

$$\mathrm{Torque} = \mathrm{K}(\theta_{\mathrm{d}} - \theta_{\mathrm{r}}) - \mu\theta' \qquad (3.36)$$

where θ_d is the desired angle, θr is the current angle, θ' is angular velocity, K is the gain of the proportional term, and $-\mu$ is the gain of the derivative term.

At this point there are two initial torques, one thigh orientation, and two landing joint angles (or two more joint torques, since they are interchangeable), for a total of five control parameters. Some "good" jumps generated through a trial-and-error process have shown that the described control parameters are sufficient to allow the robot to jump. In fact, in some situations, this set of control parameters contains redundancy, leading to multiple solutions to the control problem.

3.7 Angular Momentum and Its Significance

The variation of the thigh orientation supplies flexibility in positioning the mass center at the beginning of a jump. This is one way to adjust the angular momentum, which is determined as the integral of the product of moment about the center of mass, and time:

$$H = \int (F_x(t) \cdot L_x(t) - F_y(t) \cdot L_y(t))dt \qquad (3.37)$$

Figure 3.5 illustrates the physical meaning of the variables in equation (3.37).

As a description of the rotational motion of a multibody, angular momentum, by definition, is the product of the moment of inertia and the angular velocity about the same axis. In the case where angular momentum is constant, i.e., in the ballistic phase, increasing the moment of inertia results in a decrease in angular velocity, and vice versa. The performance of an ice skater provides an explanation of this principle. A skater turns faster when he decreases his moment of inertia by drawing his legs and arms close to the body. Conversely, he turns slower when he stretches out his legs and arms. It should be noted that for the problem at hand, the range of variation in moment of inertia is limited, therefore the variation in body angular velocity is also limited.

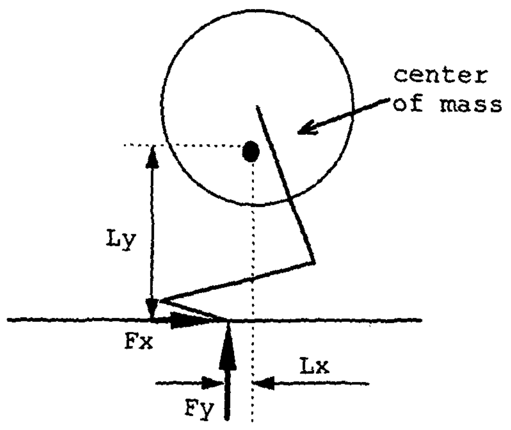

Figure 3.5 Factors of Angular Momentum

SECTION 4

Neural Networks

4.1 Multilayered Neural Networks with Backpropagation Learning

In general, a neural network consists of highly interconnected processors called neurons which perform very simple operations. In a multilayered feedforward neural network, the processors are logically arranged in two or more layers, an input layer and an output layer, each containing at least one neuron. Usually, there are also hidden layers located between the input and output layers. A neuron receives outputs of neurons in the preceding layer and combines them in a weighted sum. Figure 4.1 shows the structure and principle of a single neuron, where *f(x)*, called the activation function, provides a nonlinear mapping of the input sum to a fixed-range output.

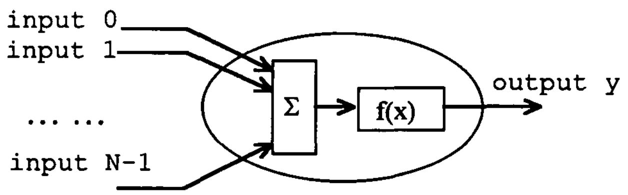

Figure 4.1 Schematic of a Neuron

The output of each neuron is a function of the weighted sum of that neuron's inputs. A neuron in the input layer, which does not have a preceding layer, simply transmits the input value unaltered. In addition to the *N* inputs, a neuron also has a *bias* signal. The neuron's output y_j is expressed as

$$y_j = f\left(\sum_{i=0}^{N-1} x_i w_{ji} + b_j\right) \qquad (4.1)$$

where $\{x_i,\ i = 0, 1, \ldots, N-1\}$ are inputs, $\{w_{ji},\ i = 0, 1, \ldots, N-1\}$ are synaptic weights, and b_j is the bias of neuron j.

The activation function can be linear or nonlinear. A traditional and popular activation function is the logistic function

$$f(x) = \frac{1}{1+e^{-x}} \qquad (4.2)$$

where x represents the weighted sum of the neuron outputs from the preceding layer as computed by the neuron.

For a multilayered neural network, network inputs are passed to each neuron in the first layer, called the input layer. The output of the input layer is processed by the subsequent hidden layer and passed on to the next layer. In this way the signals are processed until the output layer generates the network outputs. The activations of all output-layer neurons are computed in one deterministic pass. Figure 4.2 illustrates the hierarchy of a three-layer neural network.

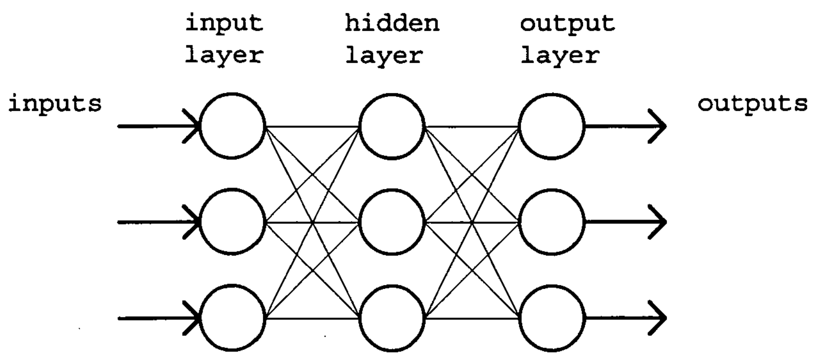

Figure 4.2 A Three-Layer Neural Network

The nonlinear mapping of a neural network depends on both the activation function and the weight matrix. The training algorithm of a neural net is then the adjustment of weights subject to certain rules. The widely used backpropagation algorithm provides a gradient-descent-based procedure for training a multilayered network. The network is first initialized with random weights and the input signals are fed through the network. The output errors, defined as the differences between the desired output values for the input, and the actual network output values, are typically large for a randomly initialized network. The errors are backpropagated through the network to correct the weight values so that output errors are reduced. When the input-output pattern p is presented, the error is computed as:

$$E_p = \frac{1}{2}\sum_{i}^{N-1}(t_{pi} - o_{pi})^2 \qquad (4.3)$$

where t_{pi} represents the *i*th component of the desired output vector, while o_{pi} is the actual output generated by the *i*th output neuron. The index *i* ranges on the number of neurons in the output layer. The total error for all input-output patterns is:

$$E = \sum_{p} E_p \qquad (4.4)$$

To perform a gradient descent in *E*, it is sufficient to correct the elements of the weight matrix by using the following rule (Masters, 1993):

$$\Delta w_{ji} = \eta \delta_{pj} o_{pi} \qquad (4.5)$$

where η is a real constant, called the learning rate, which determines the rate at which the weight matrix changes; δ_{pj} is the error due to the pth pattern, associated with the jth neuron; and o_{pi} is the output of the *i*th neuron in the preceding layer of the given neuron, when the *p*th pattern is processed by the neural network. When the learning phase is implemented, the backpropagation algorithm minimizes the square of the differences between the desired and the actual network output values summed over the output neurons and all pairs of input-output training patterns.

It has been proven that when a backpropagation network trained to model ordinary differential equations is given a proper learning rate, the training process will converge (Kuan and Hornik, 1991). However, an analytical result of a study of gradient-based learning, which is the basic learning rule of backpropagation, has shown that this learning algorithm is difficult and complex (Wang and Malakooti, 1993). The studies performed by Eaton and Oliver (1992) and Sundararajan et al. (1993) show the relationship between learning coefficients and the training data size.

4.2 CMAC Networks

In the mid 1970's, a new type of artificial intelligence technique called Cerebellar Model Articulation Controller (CMAC) was proposed (Albus, 1975). However, it was not until the late 1980's that researchers began to apply the CMAC network to control problems. Unlike neural networks in a narrow sense, the CMAC contains no neurons. Instead, the CMAC models the mathematical concepts of how the cerebellum structure inputs data, how it computes the addresses where the control signals are stored, how the memory is organized, and how the output control signal is generated. The CMAC also has the properties desired for an intelligent controller, such as incremental learning, generalization, nonlinear mapping, and parallel processing.

In fact, CMAC is a table look-up algorithm which models nonlinear functions, *f(s)*, where *s* is a discrete input state vector of dimension *N*. This computation scheme is illustrated in Figure 4.3. From the left, each input point in the input state space *S* maps to *C* locations in associative memory *A*.

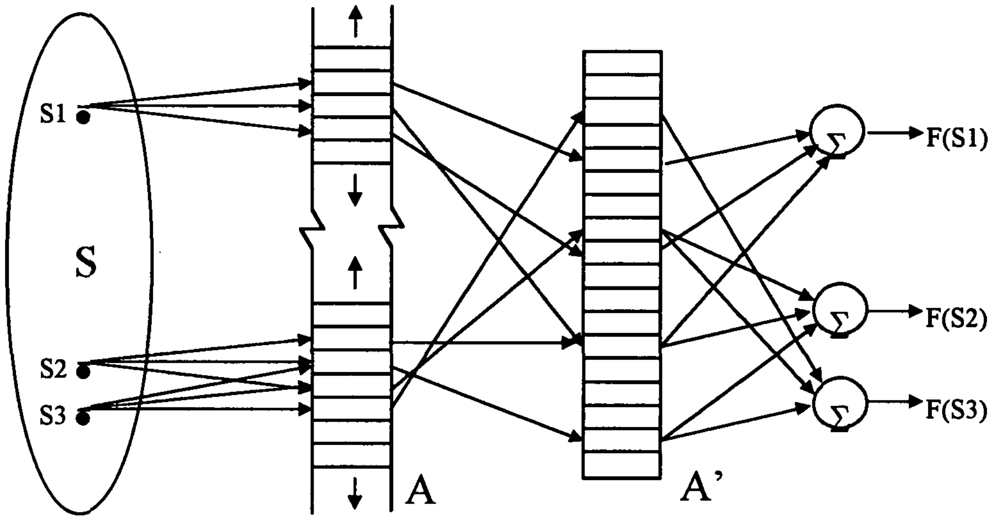

Figure 4.3 CMAC Mapping

The constant *C* is called the generalization parameter. In a basic mapping procedure, the output of the CMAC is generated by summing the values of each location in *A*. In practical problems, the size of *A* is usually too large to be implemented, even for a problem of very small size. For example, a three-input problem, with each input element divided into 100 units, has the size of *A* at 100^3, or a total of 10^6 locations! On the other hand, the number of points in *S* encountered in a practical control problem is typically much smaller. Therefore, a uniform random mapping of logical memory A into a smaller physical memory A', called hashing, was implemented. After the $A \rightarrow A'$ hash coding procedure, the output of the network is computed by summing the values of the C cells in the physical memory A'.

The size of A', expressed as $L_{A'}$, should be chosen properly such that it is not too large for a common computer, while at the same time it is able to contain all the cells mapped from A. Given a good hashing function, $L_{A'}$ can be selected such that mapping collisions are limited. When collisions occur, they do not decrease the designed generalization ability, but rather yield undesired generalization among points in the state space which are not in the same neighborhood. The effect of collisions is identical to the existing problem of learning interference, which is handled by iterative data storage (Albus, 1975). The undesired generalization between distant points can be overcome by properly selecting C in such a way that $C << L_{A'}$. When $C < 0.01L_{A'}$, the effect is very slight. When $C < 0.001L_{A'}$, the undesired overlap is practically eliminated.

Overlaps, on the other hand, are a necessary characteristic of the $S \rightarrow A$ mapping process. Each point in S corresponds to C locations in A and thus A'. Two points in A are considered far from one another when the distance between them is greater than C. They are near if the distance is smaller than C, in which case there will be an overlap between the two corresponding sets of memory locations in A. The size of the overlap depends on the distance from one point to another. The closer two points are in S, the greater the overlap. The extent of generalization within the state space is determined by the parameter *C*, as illustrated in Figure 4.3.

The nonlinear representation ability of CMAC lies in the $S \rightarrow A$ mapping, rather than depending on nonlinear activation functions as in traditional neural networks. Since the output of the CMAC is simply the linear sum of the values in the *C* locations in A', the learning process is simply to distribute the error evenly to these locations and to adjust the values in all *C* weights. Suppose that a number of training sets are available, and that each set contains an input vector and a desired output vector. Initially, when the CMAC has not been trained, its outputs are far from the required outputs. If $f(s_0)$ indicates the computed output of the CMAC and T_0 the desired output, then the correction value δ which must applied to each of the *C* weights is calculated as:

$$\delta = \beta \cdot (T_0 - f(S_0))/C \tag{4.6}$$

where β is the learning rate, which ranges from 0 to 1. This is known as the least-mean-square (LMS) rule. Each weight is incremented by δ. Usually, β is smaller than 1, and the training process will be a repeated weight adjustment algorithm. When the maximum error falls below a specified tolerance level, the network is considered trained.

CMAC is more and more used in controls engineering because of its capabilities in local generalization, nonlinear representation, and fast learning. The interpolation limitations of CMAC are analyzed by Brown and his coworkers (Brown et al, 1993).

SECTION 5

System Architecture and Approaches to Uniped Control

5.1 The Biped Controller

Figure 5.1 shows the multilayered architecture controller for a biped robot. At the lowest level are two uniped controllers, each controlling a single leg. The biped coordinator schedules the legs as well as compensates for the presence of two legs. The motion planner generates motion plans for the legs with inputs from a navigation plan and environmental inputs. The motion plans are transmitted to the appropriate uniped controller via the biped coordinator. Finally, the navigator generates navigation plans based on locomotion objectives.

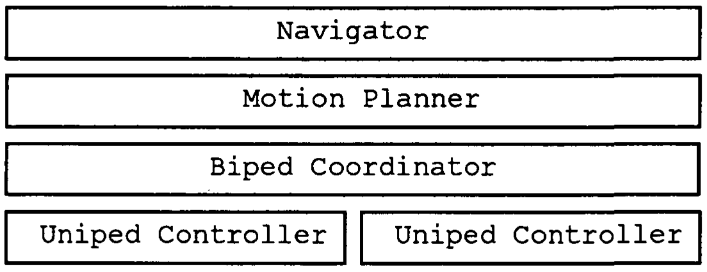

Figure 5.1 Biped Locomotion Controller Architecture

5.2 The Uniped Controller

A diagram of the uniped controller is shown in Figure 5.2. The control parameters are denoted as T0, T1, and Φ, where T0 and T1 are the joint torques, and Φ represents the initial orientations of links (currently only the thigh link). The jump-off controller is responsible for generating the required jumping height and distance. It also controls the in-flight body angular momentum, which becomes important when the robot plans before landing to prepare itself for the next jump. To accomplish its tasks, the jump-off controller orients the robot properly at the beginning of the jump and applies the appropriate joint torques over a fixed duration during the support phase.

The in-flight configuration controller, on the other hand, coordinates the positions of the links to avoid obstacles as well as to prepare the robot for landing. The "correct" in-flight body configuration is generated by the motion planner as a series of desired position configurations at discrete points during flight. The desired configurations at other times are linearly interpolated among these discrete points. The control signals issued by the in-flight controller are the joint torques at periodic control points.

The switchover from the jump-off controller to the in-flight controller is done at the instant the foot leaves the ground completely. This can be accomplished by a simple gating controller.

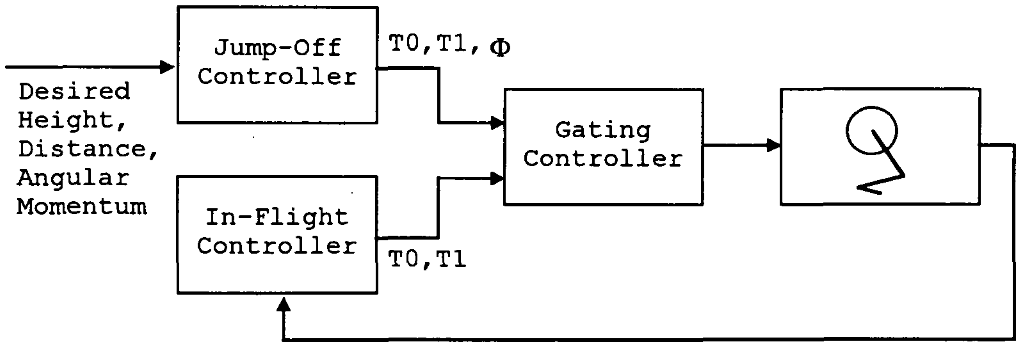

Figure 5.2 Uniped Controller Architecture

5.3 Approaches to Uniped Control

This study investigates several approaches to uniped control. Section 6 investigates various neural network approaches, Section 7 investigates a self-organizing multiagent cerebellar model, and Section 8 investigates a modular fuzzy-neural approach. The approaches are summarized here and described in detail in the following sections.

The first approach in Section 6 uses a single back propagation neural network to learn both the three jump-off control signals and one set of in-flight 'swing torques' which are applied at the top of the trajectory. The objective is to produce jump-off signals which will achieve the desired height, distance and angular momentum and at the same time produce swing torques which will result in a proper landing configuration. The second approach in Section 6 uses a CMAC neural network for the jump-off controller and a separate PD controller for the in-flight stage. The control signals issued by the PD controller are joint torques at periodic control points along the in-flight trajectory. Section 7 proposes a new self-organizing multiagent cerebellar model which is targeted to learn the control signals to achieve the objectives of both the jump-off and in-fight stages. In-flight signals are generated for one or possibly more control points along the trajectory. Section 8 uses a modular approach with separate controllers for the jump-off stage and in-flight stage. A self-organizing fuzzy-neural controller is used for jump-off control, and a CMAC-based controller is used for in-flight control. The in-flight controller learns control signals for each time step in the trajectory.

SECTION 6

Experiments in Neural Network Uniped Control

6.1 Training of BP Neural Networks

A three-layered backpropagation neural network was built as the uniped controller for two purposes: (1) to develop the appropriate topology; and (2) to determine the appropriate training parameters. The numbers of neurons in the input and output layers are determined by the problem, while the number of hidden neurons is determined by empirical rules or by trial-and-error. The rule-of-thumb for the selection of hidden neurons is that the more complex the input-output mapping is, the more neurons should be used (Masters, 1993).

At the beginning of the investigation, the neurocontroller was trained with several algorithms. Following the traditional training algorithm, an attempt was made to generate a number of "good" jumps by means of a trial-and-error procedure. After a number of such example jumps was generated, the training of the neural net appeared difficult. Believing that a three-layer neural net could model this mapping, and following the rule concerning the number of hidden neurons (Masters, 1993), the size of the network was increased to extend the capacity of the system, but the results were still unsatisfactory.

To avoid the cost of training a large network, the network was then broken into five smaller networks (A complete jump requires five control signals; see section 3.6.), each for one control parameter, in the hope that training would be easier and faster. Surprisingly, the results remained the same. The lack of convergence during training was traced to the high degree of complexity of the control problem. After the relationship between the control parameters and the jump quality parameters was carefully investigated, it was realized that the control problem was underconstrained, and therefore the solution was not unique. Neural networks by their nature cannot learn an inverse mapping if the inverse is not uniquely defined.

The training process was then altered to online reinforcement training, due to the fact that reinforcement learning does not require pregenerated examples but instead begins arbitrarily from a point in the control parameter space and follows a single path to the solution. The key to applying reinforcement learning is to define a performance index (PI) function to maximize, or equivalently, to minimize a cost function. For the jump-off controller, the cost function was defined as the scalar error (SE) of a jump, which was measured as a weighted Euclidean distance between the desired jump parameters and the actual values:

$$SE = \sqrt{(w_H E_H)^2 + (w_D E_D)^2 + (w_A E_A)^2} \tag{6.1}$$

where the w terms are weights applied to the errors according to their relative importance, and the E-terms represent the errors in jump height, jump distance, and angular momentum.

In order to make the problem as simple as possible, w_D and w_A were set to zero, making the jump height the single control objective. Accordingly, the subset of control parameters of interest consists of the thigh orientation and the joint torques at jump-off. The control problem is then underconstrained. The key to this approach is to start from a single point and then extend it in two opposite directions. Thus the training follows a single path instead of a multivalued function.

Normally, training a neural net with the backpropagation algorithm results in the minimization of the scalar output error (SE). In reinforcement training, the input to the neural net is the jump height, and the output is the control parameters. There are no target, or desired, control parameters available for comparison to the actual output. Theoretically, as the controller generates correct signals, the gradients of the actual jump height with respect to the signals are minimized to zero. In other words, the minimization of SE (SE=0) is equivalent to setting the gradients of SE with respect to the control parameters equal to zero:

$$\frac{\partial SE}{\partial z_i} = 0 \tag{6.2}$$

It should be noted that the gradients are computed using only local information. As such, the training algorithm searches out the first solution path and follows that path to the exclusion of others. To obtain gradient information, each control parameter is repeatedly perturbed by a small amount with the gradients approximated as follows:

$$\frac{\partial SE}{\partial z_i} \cong \frac{\delta SE_i}{\delta z_i} \tag{6.3}$$

where δz_i is the perturbation in control parameter z_i and δSE_i is the resulting variation in *SE*. Four simulations are required to generate the gradients at a single point. This training algorithm is shown in Figure 6.1.

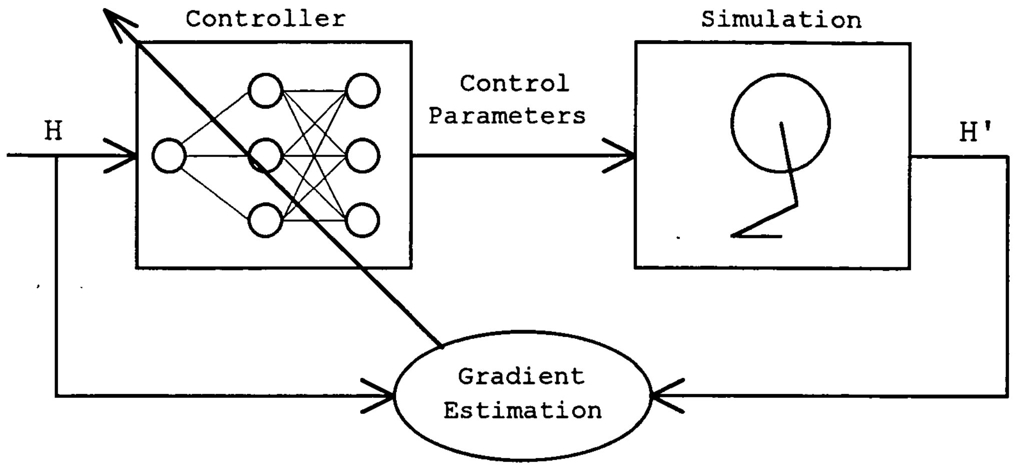

Figure 6.1 Reinforcement Training Algorithm

Training results for two different network topologies are shown in Table 6.1. The first two controllers were trained for a single jump height. The last controller was trained for three jump heights.

Table 6.1 Reinforcement Training Results

Controller	# Hidden Neurons	Jumping Height(s) (m)	Final Error
1	15	0.9	0.0890
2	20	0.9	0.0017
3	20	0.7, 0.9, 1.1	0.0780

After training, controller 3 was used to generate two jumps with desired heights of 0.868 m and 1.157 m. The results are listed in Table 6.2. Figure 6.2 illustrates the robot dynamics during the generated jumps. These are in fact "half" jumps because the only goal is the jump height.

Table 6.2 Test Jump Parameters and Results Comparison

Desired Height (m)	Actual Height (m)	Error (%)
0.868	0.901	3.7
1.157	1.243	7.4

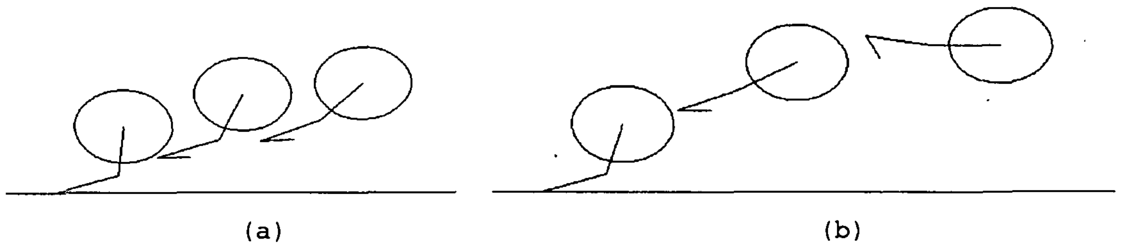

Figure 6.2 Jumps Generated by Trained Neurocontroller

(a) Desired Height = 0.868 m, Actual Height = 0.901 m

(b) Desired Height = 1.157 m, Actual Height = 1.243 m

Although training results show that the gradient-based reinforcement training with one object works reasonably well, further experiments showed that this training procedure does not scale up well for multiple control objectives. For a non-zero W_D (inclusion of jump distance as the second control objective), it was observed that once the error is reduced to a certain level, the errors E_D and E_H increase and decrease alternately, resulting in oscillations in SE about an unacceptably large value. The reason for these oscillations is that with respect to some of the control parameters, the gradient of H is positive while the gradient of D is negative, and vice versa. This is the principal difficulty in applying gradient-based reinforcement learning to multiobjective control. Another drawback is that this training algorithm requires the running of the robot simulation several times in each iteration, which is very time consuming. A large scale multiobjective problem is highly impractical.

Therefore, off-line training was resumed. A further analysis of the jump procedure revealed that if a jump is divided into phases associated with separated controllers, as presented in section 3.5, then the training of a jump-off controller for the first phase requires only three control parameters. However, this training procedure, even though it leads to convergence, also proved too time consuming to be practical. At this point, the research was directed to searching for a more suitable learning technique.

6.2 Training of CMAC Networks

After further analysis of the control problem, a new control strategy was developed which uses a CMAC controller for the jump-off phase of the stride and a PD controller to control the movement of the leg during flight. The PD controller is described in Section 3.6. A CMAC network package was developed to be trained as the uniped jump-off controller. A set of jump examples was generated which includes three control parameters corresponding to the two jump torques and the orientation of the thigh link, and three kinetic parameters corresponding to jump height, jump distance, and body angular momentum. In order to reduce the time spent on generating examples, the three kinetic parameters were computed once the robot left the ground, rather than obtaining them from simulations of the entire jump. It was reasoned in Chapter 3 that the jump height and angular momentum can be calculated exactly in this way, while the computed jump distance might be slightly different from the actual jump distance. In general, jump requirements will emphasize jump height and angular momentum, while ignoring slight discrepancies in the jump distance. The three desired jump parameters and the three control signals form a three-to-three mapping, with no redundancy. A heuristic proof is based on the observation that jump height, jump distance and angular momentum are orthogonal to one another, and that the three control signals are also independent.

Three training runs were conducted with CMAC generalization parameters (expressed as *C* in section 4.2) of 160, 320 and 480. After training, the controller was tested with a point in the input space not included in the training set. Table 6.3 lists the kinetic performance for the test cases. It is clear that the generalization parameter *C* plays an important role in CMAC training. After 200 iterations, for *C*=160, the maximum relative error was 8.3% for the body angular momentum parameter. For *C*=320 and *C*=480, the maximum relative error became 2.3% and 2.2%. Training was also very fast; 200 iterations took only minutes on a 50 MHz 486 computer. Further training improved accuracy for the case of C=160 to 6.4%, while it did not improve for the other two cases.

Table 6.3 CMAC Training Results

	Jump Height	Jump Distance	Angular Momentum	Maximum Error	Iterations Used
Desired	0.820	0.500	0.500		
C=160	0.819	0.510	0.532	8.3%	200
C=320	0.819	0.500	0.511	2.3%	200
C=480	0.821	0.498	0.489	2.2%	200

6.3 A Comparison Between CMAC and BP Neural Networks

Training runs similar to those described in the previous section were performed on a CMAC network and a BP neural network with 30 hidden neurons. CMAC training with 400 iterations required several hours on a 50MHz 486 computer to reach convergence. In contrast, the BP network took more than 20,000 iterations, lasting two days on the same machine, to achieve convergence. Table 6.4 lists the performance comparison for two test jumps. It is obvious that the performance of the CMAC is superior to that of the BP network both in terms of training time and training accuracy. Thus the conclusion was reached that CMAC is more suitable for locomotion control.

Table 6.4 Comparison of Performance Between Backpropagation and CMAC ($C=160$)

	Jump Height	Jump Distance	Angular Momentum	Maximum Error	Iterations Used
Desired	0.920	0.500	0.500		
BP	0.920	0.489	0.567	13.4%	5,000
BP	0.916	0.506	0.498	1.2%	20,000
CMAC	0.920	0.503	0.474	5.6%	33
CMAC	0.920	0.502	0.502	0.4%	433
Desired	0.950	0.600	0.480		
BP	0.948	0.609	0.496	3.3%	20,000
CMAC	0.945	0.588	0.483	2.0%	433

6.4 CMAC Generalization Parameter and Hash Coding

The total number of weights utilized in a mapping also depends on the generalization parameter *C* and the method of selecting the C weights to be updated for generalization purposes. We use the original method proposed by Albus (Albus, 1975). We discovered that this method results in an interesting anomaly. In the normal scale of *C* (*C* is smaller than the number of discrete points in one coordinate of the input space), the larger *C* is, the more overlap there is in the mapping and the fewer will be the number of regarded locations in the associative memory. This is in direct opposition to intuition. As an example, for a simple two-input problem with a resolution of 1.0 and a range of [0,5] for both inputs, with *C*=1, *C*=2, and *C*=5, the *S*-to-*A* mapping and logical memory address are listed in the following tables (Table 6.5, 6.6 and 6.7). These tables give different mappings to the logical addresses of the values 0, 1, 2, 3, and 4 in both of the two input elements. Table 6.8 lists the combinations of these addresses in the locations in the associative memory. In Table 6.8, the addresses from 00 to 44 (in total, 25 locations) are used when *C*=1; addresses 00, 02, 04, 11, 13, 15, 20, 22, 24, 31, 33, 35, 40, 42, 44, 51, 53, and 55, (in total, 18 locations) are used when *C*=2; addresses 00, 05, 11, 16, 22, 27, 33, 38, 44, 50, 55, 61, 66, 72, 77, 83 and 88 (in total, 17 locations) are used when *C*=5. As C gets larger, the

We have recently found in the literature a method which selects for update weights which are more evenly distributed (Parks and Militzer 1991). Future studies should investigate whether the Parks and Militzer method provides more accurate generalization.

Table 6.5 Mapping from Input to Logical Address (*C*=1)

Value	0	1	2	3	4	5	6	7	8	Logical Addresses
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	2
3	0	0	0	1	0	0	0	0	0	3
4	0	0	0	0	1	0	0	0	0	4

Table 6.6 Mapping from Input to Logical Address (C=2)

Value	0	1	2	3	4	5	6	7	8	Logical Addresses
0	1	1	0	0	0	0	0	0	0	01
1	0	1	1	0	0	0	0	0	0	21
2	0	0	1	1	0	0	0	0	0	23
3	0	0	0	1	1	0	0	0	0	43
4	0	0	0	0	1	1	0	0	0	45

Table 6.7 Mapping from Input to Logical Address (C=5)

Value	0	1	2	3	4	5	6	7	8	Logical Addresses
0	1	1	1	1	1	0	0	0	0	01234
1	0	1	1	1	1	1	0	0	0	51234
2	0	0	1	1	1	1	1	0	0	56234
3	0	0	0	1	1	1	1	1	0	56734
4	0	0	0	0	1	1	1	1	1	56784

Table 6.8 Addresses in the Associative Memory

Inputs	C=1	C=2		C=5				
00	00	00	11	00	11	22	33	44
01	01	02	11	05	11	22	33	44
02	02	02	13	05	16	22	33	44
03	03	04	13	05	16	27	33	44
04	04	04	15	05	16	27	38	44
10	10	20	11	50	11	22	33	44
11	11	22	11	55	11	22	33	44
12	12	22	13	55	16	22	33	44
13	13	24	13	55	16	27	33	44
14	14	24	15	55	16	27	38	44
20	20	20	31	50	61	22	33	44
21	21	22	31	55	61	22	33	44
22	22	22	33	55	66	22	33	44
23	23	24	33	55	66	27	33	44
24	24	24	35	55	66	27	38	44
30	30	40	31	50	61	72	33	44
31	31	42	31	55	61	72	33	44
32	32	42	33	55	66	72	33	44
33	33	44	33	55	66	77	33	44
34	34	44	35	55	66	77	38	44
40	40	40	51	50	61	72	83	44
41	41	42	51	55	61	72	83	44
42	42	42	53	55	66	72	83	44
43	43	44	53	55	66	77	83	44
44	44	44	55	55	66	77	88	44

Another important factor affecting CMAC performance is hash coding. The purpose of hashing is to minimize the use of physical memory so that the implementation of CMAC becomes practical. Hashing is essentially a many-to-few mapping. Unlike a general purpose random number generator which generates a sequence of pseudo-random numbers given a seed, a hashing function generates uniformly distributed random numbers in a given range, fed with seeds distributed in any way in the input space. Table 6.9 shows the results of distribution quality tests conducted for the hashing function used in CMAC implementation. The term "times" represents the number of times a point appeared in the output space during the mapping from the given seed (input) space. This table shows that the quality of the hashing function is usable, even though not ideal, especially in mappings from a large space to a space with a size of about 10,000.

Table 6.9 Quality Test of the Hashing Function

Seed Space Size	Output Space Size	Maximum Times	Minimum Times	Mean
10,000	10000	2	0	1
100,000	10000	13	8	10
1,000,000	10000	127	84	100
10,000,000	10000	1226	908	1000
100,000	1000	103	97	100
1,000,000	1000	1014	972	1000

Comparisons were made between straight access and hashing access to weight memory in CMAC training. Table 6.10 shows the result after learning a four-fold parity using two different accesses. In the case where the memory size is sufficiently large, there was no difference in the number of iterations; however, when memory size was reduced so that hashing became necessary, the number of iterations required to reach a given accuracy increased. This experiment proves that the cost of using hash coding is in learning accuracy and training time.

Table 6.10 Results of Learning Parity with Two Accesses

	C	Size of Weight Space	Learning Rate	Iterations Used	Max Error
Straight	4	10000	0.6	7	0.000983
Hashing	4	10000	0.6	7	0.000983
Hashing	4	100	0.6	18	0.000911

The results of learning the function *z=sin(x+y)* with different sizes of physical memory, listed in Table 6.11, reveal that it is improper to select a memory size larger than necessary ($L_{A'}$=5000). Also, it is seen from this table that learning accuracy, limited by collision, becomes unacceptably low when memory size is too small ($L_{A'}$=500).

Table 6.11 Results of Learning the Function *z=sin(x+y)*

Generalization Parameter *C*	Size of Weight Memory	Learning Rate	Iterations Used	Maximum Error
5	5000	0.6	17	0.009424
5	1000	0.6	17	0.009424
5	500	0.6	200	0.012200
5	500	0.6	300	0.012210

6.5 Conclusions

The CMAC network has exhibited some important and beneficial features during the experiments performed in this study. Specifically:

1. It has the ability to learn arbitrarily nonlinear functions without the mathematical expressions of the function;
2. CMAC training is much faster than BP training because the number of computations grows only linearly with the number of system state variables, as shown in section 6.3;
3. Since the CMAC is essentially a kind of lookup table, a hashing technique can scale a large input space associated with complex problems down to a practical physical memory space;
4. The measure of local generalization is adjustable through variation of that generalization factor. In regions of the input space which are relatively linear, interpolation may still suffice for very sparse data sets, provided the generalization factor is selected properly.

The jump-off controller is used with three inputs corresponding to the desired jump height, jump distance, and body angular momentum, and three outputs corresponding to the two joint torques and the orientation of the thigh link. The link configuration controller has six inputs corresponding to the sizes, velocities and accelerations of the two joint angles, and two outputs corresponding to the two torques.

Although this study deals strictly with the uniped controllers at the bottom of the hierarchy, the uniped controller is currently being trained and fine-tuned for single jumps responding to jump behavior requirements within a specified range.

With respect to continuous jump control, additional effort is needed to investigate the control strategy of the transition phase, which is the connection of one step to the next. As analyzed in Chapter

3, the quality of a continuous jump depends on the landing configuration and the properly selected angular momentum for take-off according to the stipulations for jump distance and jump height.

Instead of the simple PD controller used in this section, an intelligent controller is necessary for the ballistic phase so that preconditioning of the transition phase will be more controllable. To develop a ballistic controller using the techniques of this section a prohibitively large amount of data representing the performance of the robot in air is needed for training. The next two chapters develop alternative techniques which reduce or eliminate the need for precomputed training examples.

SECTION 7

Mac-J: A Self-Organizing Multiagent Cerebellar Model For Fuzzy-Neural Control Of Uniped Locomotion

This section describes a novel approach for developing a fuzzy-neural controller for both the jump-off and in-flight phases of uniped locomotion. The major contribution of this work includes:

(1) the one-to-many cardinality was determined between desired jump measures and jump control parameters for full-degree take-off freedom and full jumps; and

(2) a self-organizing multiagent cerebellar model, MAC-J, is proposed and developed such that it can learn and control full jumps with full degree take-off freedom.

MAC-J is unique in two aspects. Theoretically, it introduces multiagent distributed AI (DAI) concepts into intelligent control and presents a coordinated computational intelligence (CCI) approach to legged locomotion. Technically, it bridges a gap between complex motion equations and high dimensional fuzzy-neural control with four common sense "cerebellar" laws; it emulates human and animal locomotion learning with associative memory aggregation and reorganization; and it is generic, time- and storage-efficient, and ideal for microelectronics design/manufacturing. MAC-J has been implemented in software and was successfully tested with four different uniped simulations. Test results show that, starting with a given jump example (assisted jump), MAC-J can enable a uniped to learn different jumps quickly with a learning, tuning, and brainstorming process. The result is being evaluated by NASA-JSC for a possible patent.

Technical details of MAC-J have been presented at the group meeting in May with NASA personnel and fully reported to the Automation and Robotics Division at NASA Johnson Space Center. Since this work is being evaluated for a possible patent by NASA, only general ideas are discussed in the following:

Step 1. Agent-oriented decomposition:

Corner parameters are identified in this step to achieve a one-to-one mapping based on the three take-off configuration parameters: q1, q2, and q3. Given initial (temporary) minimum and maximum values for qi, i = 1,2,3, corner agents are identified. A corner agent is a fuzzy-neural controller responsible for locomotion control in its subspace, named a corner. A cornered-space is a subspace within the originally unknown control space. Any pair of adjacent agents in a cornered-space must meet the certain neighborhood conditions.

Step 2. Agent tuning with learning by practicing:

Each individual agent consists of a lookup table associative memory [Albus, 1975], an indirect fuzzy controller interface [Brown and Harris, 1994], and a BP (backpropagation) neural network [Rumelhart and McClelland, 1986]. The numerical associative memory is for efficient adaptive and incremental learning, the fuzzy controller is for linguistic inversion of the associative memory and interface purposes, and the BP neural net (optional) can be used for fuzzy-set fine-tuning. The fuzzy controller is used as a local interface due to its "white-box" property (each local agent needs to know its functional limits). To meet the neighborhood conditions, associative memory elements are fine-tuned with two converging functions. The two equations form a key for a learning by practicing process. Given a single jump example, the process can generate and fine-tune associative memory elements to desired similarities for all eight corner agents. It should be remarked that, with the agent-oriented approach, the number of examples required by each agent is extremely small. With a small number of fine-tuned examples, a fuzzy controller [Wang and Mendel, 1992], a CMAC controller [Albus, 1975], or a BP neural net [Rumelhart and McClelland, 1986] can be trained easily (need seconds or minutes). This simplification is compensated by a powerful brainstorming process in the next step.

Step 3. Agent discovery with learning by brainstorming:

Brainstorming is defined as a high-level learning process where new agents are discovered collectively by existing corner agents. Learning by brainstorming leads to cerebellar memory booming which provides a partial explanation to the fact that children and baby animals learn locomotion control rapidly after their first few steps.

Step 4. Cerebellar law discovery:

Memory booming can not go on forever. Then arise the questions: Is agent discovery the highest level of learning? How many cerebellar agents are needed for usual locomotion functionalities? How are the agents self-organized? To answer these questions, the notion of a safe memory space is defined. A safe memory space can grow (leading to expanded locomotion abilities when the body "grows up") or shrink (leading to reduced locomotion capabilities when memory is "damaged"). Based on the safe space definition, four cerebellar motion laws are discovered and self-verified by MAC-J. The minimum number of agents required for a safe space is determined as 2^N, where N is the number of links of a uniped robot.

Step 5. Cerebellar memory expansion and agent reorganization:

The significance of the cerebellar laws is 3-fold: 1) they bridge a gap between motion equations and multiagent fuzzy-neural control by providing common sense inverse dynamics; 2) they limit the number of cerebellar agents needed for usual locomotion functionalities; and 3) they provide a theoretical basis for the aggregation and reorganization of cerebellar agents in locomotion learning. With the theoretical basis, MAC-J is extended to a five-layer cerebellum architecture with a virtual memory scheme and an efficient computational basis for an intelligent microprocessor technology. It is intelligent in the sense of being able to learn and control uniped robots with different numbers of links, changing sizes, and varying weights.

Step 6. Multiagent locomotion control with cooperation and competition:

With the cerebellar laws, uniped locomotion learning and control can be efficiently performed online by the corner agents in a safe space based on a fuzzy coordination protocol.

MAC-J is unique in two aspects. Theoretically, it introduces multiagent coordination techniques of distributed AI (DAI) [Bond and Jasser, 1987; Zhang, 1996; Zhang, 1992] into intelligent control and presents a coordinated computational intelligence (CCI) approach to legged robot locomotion that can emulate adaptive behaviors in human and animal locomotion learning processes. Technically, it bridges a gap between complex motion equations and high dimensional fuzzy-neural control; it supports "cerebellar" memory expansion and reorganization; and it is generic, time- and storage-efficient, and ideal for microelectronics design/manufacturing.

It is recommended that follow-on work be implemented by NASA to further explore the benefits of his novel approach.

SECTION 8

A Modular Neuro-Fuzzy Approach to Uniped Control

In this section we investigate a modular neuro-fuzzy approach for the uniped controller. We use the modular architecture presented in Figure 5.2, with separate controllers for the jump-off and in-flight stages of the running stride. This decoupling is a natural one because the control parameters and control objectives for the two stages are different. The jump-off controller produces the torques and thigh orientation which are applied to the joints at takeoff in order to achieve the desired height, distance and angular momentum of the stride. After the foot leaves the ground, control is switched to the in-flight controller. It takes the current state of the links (angular position, velocity and acceleration) and produces the set of torques targeted to move the links smoothly to the next position.

There are two important advantages to the approach taken here. First, it is simple. Second, it allows the leg to be controlled at each time step during the trajectory, rather than at one or more isolated points. This is critical for achieving smooth movement of the leg during flight and for avoiding obstacles.

Designs of the jump-off and in-flight controllers are presented in the following sections, along with initial results obtained thus far using this approach.

8.1 The Jump-Off Controller

For the highly non-linear jump-off control problem we require a controller which is adaptive, trains quickly and accurately and can be trained on-line. We select a fuzzy-neural controller which combines learning techniques from self-organizing neural networks and weighting techniques borrowed from fuzzy logic membership functions. It is based on an approach has been successfully used to train a truck to back up to a loading dock (Kwon, 1994).

The fuzzy-neural controller uses locally-tuned receptive fields, updating only local information for each training pattern. This makes learning fast and also permits incremental on-line learning, since changes in portions of the problem space will not require off-line retraining of the entire space. This permits gradual growth of the problem space and/or relaxation of the simplifying constraints mentioned earlier without global retraining.

To train the controller, equations of motion governing the movement of the robot are used to generate examples. Each training example consists of an input state/target control signal pair. The input state space S in our problem is three-dimensional and corresponds to the desired height, distance and body angular momentum of the stride. There are three target control signals T to be learned: the torques to be applied to the ankle and knee joints and the orientation of the thigh link at takeoff.

The fuzzy-neural controller is trained by presenting it with a series of examples (S_i, T_i). The network learns by forming clusters, each of which represents a range of input states S whose corresponding target control signals T are similar. The basic principle involved is that similar control signals will be used to produce similar results from similar stating states. Each cluster is represented by a prototype (Sp, Tp) pair where the prototype input state Sp represents the center of the cluster of input states and the prototype target control signal Tp is the set of control signal values corresponding to this cluster of inputs. New prototypes are added as needed to achieve the desired accuracy level. When presented with an input state S_i, the network produces a response which is a weighted average of the control signals Tp corresponding to the learned state clusters Sp which are closest to S_i.

During training the following is repeated for each example (S_i,T_i) in the training set:

1. present next input pattern S_i to the controller

2. calculate the response T_i*.

3. apply control signals T_i* to the robot and determine the error E in distance, height and angular momentum

4. if E is larger than an acceptable level ε, create a new prototype positioned at the location of S_i and store T_i as the set of torques for this cluster; otherwise, incorporate this example into the existing neighboring prototypes

The response T_i* in step 2 is a weighted average of the torques corresponding to the input clusters which are closest to the current input pattern:

$$T_i^* = \frac{\sum_{j\in N} \mu_{Sp_j}(S_i) Tp_j}{\sum_{j\in N} \mu_{Sp_j}(S_i)} = \sum_{j\in N} \mu_{Sp_j}(S_i) Tp_j$$

N represents the collection of prototype states which are in the neighborhood of the input state. The weighting factor $\mu_{Sp_j}(S_i)$, borrowed from fuzzy logic, represents the degee of membership of the input state S_i in cluster prototype Sp_j, and is computed as follows:

$$\mu_{Sp_j}(S_i) = \frac{\frac{1}{d_j}}{\sum_{k \in N} \frac{1}{d_k}}$$

where d_j represents the Euclidean distance from S_i to Sp_j. This assigns a larger weight to those prototype control signals Tp whose cluster centers Sp are the closest and hence the most similar to the current input state under consideration.

We use a dynamic measure for the neighborhood. The nearest neighbor is the prototype Sp which is closest in terms of Euclidean distance to the input state S_i. Any protytopes which are no farther than twice this distance from S_i are considered to be in the neighborhood.

In step 4, an input pattern is incorporated into the neighboring clusters by adjusting the prototypes (Sp, Tp) in the neighborhood so as to reduce the error between the target response T_i and the actual response T*, using the following update rules adapted from (Kwon, 1994):

$$\Delta Tp_j = \eta 2(T_i - T^*)\mu_{Sp_j}(S_i) \forall j \in N$$
$$\Delta Sp_j = \eta 4(T_i - T^*)\frac{S_i - Sp_j}{d_j^2 \sum_{k \in N} \frac{1}{d_k}} \forall j \in N$$

where η is a learning rate << 1. The error criterion ε in 4 can be decreased over time. Training stops when the average error falls below tolerance.

To train the controller we used the equations of motion governing movement of the robot to generate a collection of 3152 examples with a random uniform distribution over a portion of the state space corresponding to heights of 1.0 to 1.2 m, distance of 0 to 1.4 m, and angular momentum of -.05 to .5. We randomly selected 100 of the training examples to test the accuracy of the trained nets in generalizing. Six test sets were formed by adding randomly generated noise within a certain range to each of the 100 selected example input states. The amount of added noise ranged from 0 in Set 1 to (-0.1, 0.1) in Set 6.

To determine the accuracy of the trained controller, we measured the maximum testing error. The testing errror is obtained by presenting a testing state S to the trained controller, obtaining the controller's response f(S), applying these control signals to the simulated leg and then observing the error E_{test} between the desired height, distance and angular momentum and the actual height, distance and angular momentum achieved in the robot stride.

Table 8.1 shows the results achieved by the fuzzy-neural controller on the original 100 training points which were selected for testing, and on points generated by randomly adding various levels of noise to these points. We used an error criterion ε of 2% and a learning rate of .0001. Only one iteration through the training set was required to reach convergence, with a total of 728 prototypes formed. The maximum average testing error was below 0.5% for all of the noisy testing sets.

Table 8.1. Results achieved by the fuzzy-neural controller on noisy test sets.

Test Set	Added Noise	Max Ave Testing Error
1	(-0.00, 0.00)	0.495%
2	(-0.02, 0.02)	0.488%
3	(-0.04, 0.04)	0.497%
4	(-0.05, 0.05)	0.495%
5	(-0.075, 0.075)	0.469%
6	(-0.10, 0.10)	0.478%

For the sake of comparison, we trained a CMAC network like that described in Section 4.2 on the same data. Table 8.2 shows the average maximum testing error achieved by CMAC on same testing sets. We used a generalization parameter C of 160, and a learning rate of .3. We used cell sizes of .003 for the desired height, distance and angular momentum. This yielded a logical associative memory A of 5,703,703. A' was 25,000. The net was trained for 600 iterations. The maximum average testing error was below 5% for all of the noisy testing sets.

Table 8.2. Results achieved by the CMAC controller on noisy test sets.

Test Set	Added Noise	Max Ave Testing Error
1	(-0.00, 0.00)	1.6%
2	(-0.02, 0.02)	4.7%
3	(-0.04, 0.04)	3.2%
4	(-0.05, 0.05)	3.9%
5	(-0.075, 0.075)	3.2%
6	(-0.10, 0.10)	3.5%

The fuzzy-neural controller achieved more accurate results in less time and using less memory than the CMAC networks. However, the trained

CMAC net is simpler and has greater speed in the operational phase. The two models are compared in more detail in the following paragraphs.

The CMAC and fuzzy-neural controllers described here use approaches which are similar in some respects. They both form a map of the input space and store control signals associated with portions of the input space. Both use local learning; CMAC updates C cells, while the fuzzy-neural controller updates the cluster prototypes in the neighborhood. The main differences are in the method of map formation and the generalization scheme. These differences can result in large differences in performance.

The CMAC forms a static map of the input space by partitioning each input into fixed sized cells. Generalization is achieved by spreading the control signals over a fixed number C of neighboring cells and producing an output signal which is a sum of the signals in the C cells. The advantages are simplicity and computational speed. In particular, once the net is trained, the control signals for a given state can be accessed by a simple computation which does not involve a time-consuming search.

The disadvantages are the excessive memory requirements, potential for inappropriate generalization, and time to convergence. Memory requirements grow exponentially with the dimension of the input space. This makes the CMAC a poor choice for high-dimensional problems. The combination of fixed cell sizes and fixed generalization parameter C can result in poor generalization in parts of the input space which require more finely-grained control than others. Problems with generalization can result in many iterations required for training to converge (600 in this case, vs. 1 for the fuzzy-neural controller).

The fuzzy-neural controller forms its map dynamically by adding new prototypes as needed. This forms a partition of the input space in which the receptive field of each prototype can be of a different size and grows or shrinks as needed to achieve the desired level of accuracy. Thus the granularity of the control surface is tuned to the requirements of that part of the state space, as illustrated in Figure 8.1. Generalization is achieved by forming output signals which are a weighted combination of the prototype signals associated with neighboring control prototypes. The weighting is proportional to the distance of the prototype from the input state under consideration. In addition, the neighborhood size is not fixed but is a function of the closeness of neighboring prototypes. More accurate results can be expected from this scheme than from the CMAC scheme which uses a simple unweighted sum and a fixed number of cells as its 'neighborhood'. Memory requirements for the fuzzy-neural controller are limited to the number of prototypes needed to produce the desired level of accuracy and so are much less than the CMAC. Contrast the CMAC associative memory size A' of 25,000 cells in this example with the fuzzy-neural controller memory of 728 prototypes.

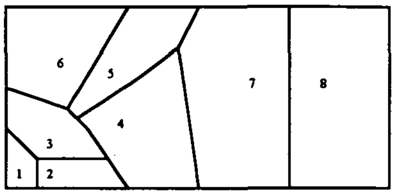

Figure 8.1. Receptive fields of fuzzy-neural prototypes Sp. The box represents a two-dimensional space S which is partitioned by 8 prototypes. The lines show the boundaries of the receptive fields of each prototype, marking the region of the space which is closest to that prototype.

The main disadvantages of the fuzzy-neural controller are greater complexity in the training algorithm (although it is still fairly simple) and computational cost during operation of the trained network. In particular, in order to generate the output signals, it is necessary to compute the Euclidean distance from the input state to each of the prototype states. However, this presents a problem in speed only if there are very many prototypes formed. In addition, each training step requires presenting the control signals to the robot simulation to get the achieved height, distance and angular momentum. However, this computational cost is offset by the fact that relatively few iterations are required to reach convergence.

8.2 The In-Flight Controller

The In-flight controller controls the movement of the joints from the time the foot leaves the ground after jump-off to the time it lands on the ground again. In the simplest case, it must move the joints into a configuration which will allow a safe landing. If there are obstacles, it must also position the joints so as to avoid collision.

The motion plans for the stride are formulated by the higher-level motion planner of Figure 5.1. The plan consists of a series of leg configurations ('snapshots') along a stride trajectory of a certain height, distance and angular momentum. The stride trajectory is one whose jump-off control signals have been previously learned by the jump-off controller. The snapshots are planned so as to avoid any obstacles present and to move the leg smoothly to a landing position. A simple interpolation scheme is used to plan the joint movements between time snapshots.

Like the jump-off phase, the in-flight phase also requires fast, accurate on-line learning. The state space for this part of the problem is very large. A supervised learning scheme like those previously described would require the generation of such a large number of examples that on-line learning would be infeasible. For this problem we select a controller which learns the input-output relationships of the robot from observation *without* precomputed examples. This controller, developed by Miller (Miller, 1987), has been successfully used to control the movement of a two-link robot arm. The controller is CMAC-based, but the CMAC learns from the experiences generated by a conventional PD controller instead of from precomputed examples.

For this part of the study we move to the four-link leg model illustrated in Figure 8.2. This leg is harder to control but is more humanlike and more versatile than the three-link model. It also provides a more demanding test of the current approach. The dynamics of the leg joints is described by the following:

$$\Theta''=g(\Theta, \Theta', T)$$

where Θ represents the vector of joint angles, Θ' represents the joint rotational velocities, and Θ'' the joint accelerations. The function g is an unknown nonlinear function of the joint angles, angular velocities, and the actuator torques T.

To control the joint movements the objective is to learn the inverse function:

$$T=g^{-1}(\Theta, \Theta', \Theta'')$$

where T is the vector of torques which must be applied to achieve the joint accelerations Θ'' given the current joint positions Θ and velocities Θ'. The CMAC can be applied to this problem by forming the desired state vector S_d from the vectors Θ, Θ', and Θ'' and training the CMAC to produce the response $f(S_d) = T = g^{-1}(\Theta,\Theta',\Theta'')$. The desired state S_d serves as the input to the CMAC and the targeted response $f(S_d)$ is the set of torques which will achieve the desired state.

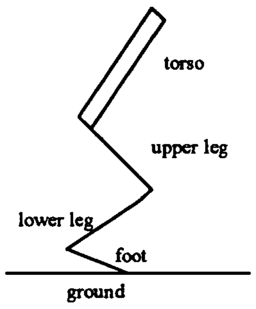

Figure 8.2. The four-link robot leg.

Torques are applied at the ankle, knee and hip joints to make the leg move.

The high-level design of this controller is shown in Figure 3. At each time step the controller proceeds as follows. The trajectory planner determines the ideal state S_i of the leg for this time step in the trajectory. This is based on the series of leg configurations which have been supplied by the motion planner. A conventional PD controller produces a set of torques which are designed to reduce the error between S_i and the observed state S_o. At the same time, the trajectory planner computes the desired acceleration A_d which will intersect the leg with the ideal state a number of time steps in the future. The desired state vector $S_d = (S_o, A_d)$ is formed from the current observed angular positions and velocities and the desired angular acceleration A_d. The CMAC is referenced to find the set of torques $f(S_d) = f(S_o, A_d)$ which will achieve the desired acceleration from the current state S_o. These torques are added to the torques computed by the PD controller, and the combined torques T are sent to the robot. This produces a new observed state S_o. At the end of the cycle, the CMAC is updated using 4.6, where T_o is the set of torques applied during the control cycle and S_o is formed from the observed positions and velocities of the joints at the beginning of the control cycle and the observed accelerations of the joints during the control cycle. Thus the CMAC learns which set of torques T achieved a certain acceleration from a certain state. If the robot encounters a similar state/acceleration goal (S_o/A_d) in the future, it will find stored in CMAC the signals which worked in the past. Initially the CMAC has no stored knowledge, and the robot is controlled exclusively by the PD controller, which is of the form shown in 3.36. Over time the CMAC learns from experience. The PD control signals decrease as the error decreases, and the CMAC takes over.

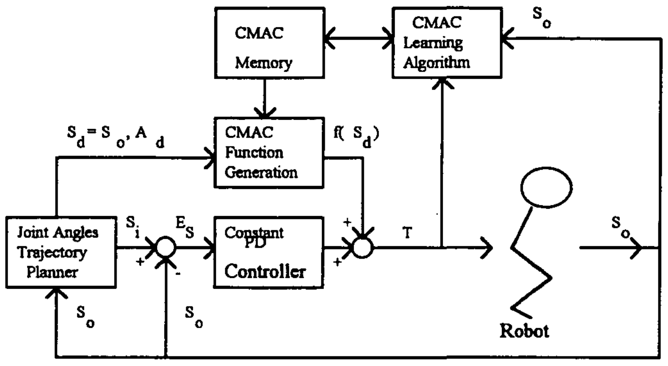

Figure 8.3. The CMAC-based controller.

We trained the in-flight controller on a motion plan formulated to fit a step trajectory of height 1.6 m, distance 0.294 m and duration 0.81s. The "ideal" joint trajectories were planned from the five snapshots shown in Table 8.3, which specify the configuration of the joint angles at five times on the trajectory: the point of takeoff, the point of landing, the top of the trajectory, midway between takeoff and the top; and midway between the top and landing. Snapshot 1 is fitted to the initial configuration of the leg at takeoff. Snapshot 5 provides a good landing configuration. Snapshots 2-4 are designed to provide clearance from the front, top, and rear of an obstacle. The number of snapshots can vary to meet the demands of the stride. Each simulated control cycle takes 0.2 ms. A simple interpolation scheme was used to obtain the ideal joint positions versus time for all the time steps between snapshots.

For the PD controller we used 1000 for the gain of the proportional term and -10 for the gain of the derivative term. We use a generalization parameter C of 100, and a learning rate of 0.2.

The input space for this problem is of nine dimensions, consisting of the observed joint angles and velocities and the desired accelerations for each of the three angles. However, since the control of each joint is independent of the others the problem reduces to three input spaces of three dimensions each. While the dimensionality is not very large, the size of the state space is. At the moment that the foot leaves the ground, the angular velocities and accelerations can be very large because large torques must be applied during jump-off in order to propel the leg off the ground. During the in-flight phase joint velocities must be small. This results in a large range of velocities and accelerations. However, the granularity of the control signals does not have to be very fine at large speeds and accelerations. Small changes in control signal are needed only to achieve small changes in angles and velocities. We therefore use a logarithmic scale for the velocities and accelerations. This greatly reduces the size of the logical memory A and still provides for fine-grained control in the part of the space where it is needed. Specifically, we use a logarithmic scale velocity range of ±12.5 degrees/s and a logarithmic scale of ±20 degrees/s^2 for acceleration. Each state variable Θ' is first converted to $\ln(\Theta')$ and then assigned to one of 200 bins, each of size 0.125 degrees/s. Each state variable Θ'' is first converted to $\ln(\Theta'')$ and then assigned to one of 200 bins, each of size 0.2 degrees/s^2. Each variable Θ is assigned to one of 180 bins of size 1 degree. Thus the logical association memory A is of size 3x(200x200x180) =21,600,000. We use a physical memory A' of size 3X72,000=216,000.

Table 8.3 Joint Angle Trajectory Snapshots.

Joint	**Snapshot Number**				
	1	**2**	**3**	**4**	**5**
ankle	106	110	115	95	70
knee	90	80	75	90	105
hip	75	80	85	98	110

To measure the performance of the controller we computed the maximum of the three errors in joint angle (ankle, knee, hip) for each time step, where the error is the difference between the actual joint angle and the ideal joint angle. We then computed the average maximum error for all the time steps in the trajectory. After only one iteration the controller learned the joint trajectories with an average maximum error of only 0.987 degree. After two iterations the error was reduced to 0.934 degree.

We are currently training the controller on additional trajectories with different height and distance characteristics.

8.3 Summary and Conclusions

This section presents a modular approach to controlling the trajectory of a single running stride. A fuzzy-neural jump-off controller controls the height, distance and angular momentum of a range of simulated running strides with an average accuracy of 99.5 percent on significantly noisy test patterns which were not included in the training set. A CMAC-based in-flight controller controls the movement of the joint angles along a planned trajectory with an average accuracy of 1 degree. Both approaches are adaptive and use local learning, which will permit on-line retraining if conditions change. The fuzzy-neural jump-off controller was trained after only one iteration through the training set, and the CMAC-based controller was trained after only two trips through the trajectory. Controllers for both phases use simple training algorithms. The CMAC-based in-flight controller is trained on the fly without precomputed examples. Because the leg joints are controlled at each time step during flight, movement is smooth and obstacles can be avoided.

Results obtained thus far demonstrate that this approach has the potential to produce fast, accurate controllers which can be trained on-line. Because the controllers use local learning techniques, they can automatically adapt to changing conditions without global retraining.

SECTION 9

Conclusions and Recommendations

This study presents a modular approach to biped locomotion control. In this approach the biped locomotion problem is reduced to the development of two identical uniped controllers whose actions are planned and coordinated by higher-level components. This phase of the research focuses on the development of a uniped controller to control a single running stride.

Several promising approaches are developed which are worthy of further investigation. The multiagent cerebellar model approach outlined in Section 7 is being considered for a patent by NASA, and details of its advantages have been described elsewhere.

The modular fuzzy-neural approach described in Section 8 is also worthy of further study. It offers the advantages of simplicity, speed of learning, and virtually continuous control of the leg movements during flight. The latter is important for smooth movement of the joints and for avoiding obstacles. In addition, the fact that the in-flight controllers learns without precomputed examples is a big advantage for this large problem space.

With respect to continuous jump control, additional effort is needed to investigate the control strategy of the transition phase, which is the connection of one step to the next. The objective for the transition phase is to smoothly move the leg from the landing configuration achieved at the end of the in-flight phase to a crouched position suitable for jump-off in the next stride. The CMAC-based approach used in Section 8 for the in-flight controller bears investigation here, since the control objectives for both phases are similar: smooth movement of the joints along a planned path to an appropriate final configuration.

References

Albus, J. S. (1975). A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC). Journal of Dynamic Systems, Measurement, and Control, Transactions of the ASME, 9, 220-227.

Albus, J. S. (1975). Data Storage in the Cerebellar Model Articulation Controller (CMAC). Journal of Dynamic Systems, Measurement, and Control, Transactions of the ASME, 9, 228-233.

Asada, H., Ma, Z.-D., and Tokumaru, H. (1990). Inverse Dynamics of Flexible Robot Arms: Modeling and Computation for Trajectory Control. Journal of Dynamic Systems, Measurement, and Control, 112, 6, 177-185.

Bond, A. and Gasser, L. (1987). Readings in Distributed Artificial Intelligence, Morgan Kaufmann.

Brown, M, and Harris, C. (1994). Neurofuzzy Adaptive Modeling and Control. Prentice Hall, New York, 218-295, 407.

Brown, M., Harris, C. J., and Parks, P. C. (1993). The Interpolation Capabilities of the Binary CMAC. Neural Networks, 6, 429-440.

Eaton, Harry A. C. and Oliver, T. L. (1992). Learning coefficient dependence on training set size. Neural Networks, 5, 283-288.

Fukuda, T., Shibata, T., Tokita, M. and Mitsuoka, T. (1992). Neuromorphic Control: Adaptation and Learning. IEEE Transactions on Industrial Electronics, 39, 6, 497-503.

Fukuta, T., and Shibata, T. (1992). Theory and Applications of Neural Networks for Industrial Control Systems. IEEE Transactions on Industrial Electronics, 39, 6, 472-489.

Furusho, J. and Sano, A. (1990). Sensor-Based Control of a Nine-Link Biped. The International Journal of Robotics Research, 9, 2, 83-98.

Goddard, R. E., Zheng, Y. F., and Hemami, H. (1992). Control of the Heel-Off to Toe-Off Motion of a Dynamic Biped Gait. IEEE Transactions on Systems, Man, and Cybernetics, 22, 1, 92-102.

Gullapalli, V, Franklin, J. A., and Benbrahim, H. (1994). Acquiring Robot Skills via Reinforcement Learning. IEEE Control Systems, 2, 13-25.

Handelman, D. A., Lane, S. H., and Gelfand, J. J. (1990). Integrating Neural networks and Knowledge-Based Systems for Intelligent Robotic Control. IEEE Control Systems Magazine, 4, 77-86.

Hodgins, J. K. and Raiber, M. H. (1990). Biped Gymnastics. The International Journal of Robotics Research, 9, 2, 115-132.

Hodgins, J. K. and Raibert, M. H. (1990). Robot gymnastics. Int'l J. of Robotics Research, 9, 2, 115-132.

Ishiguro, A., Furuhashim T., Okuma, S., and Uchikawa, Y. (1992). A Neural Network Compensator for Uncertainties of Robotics Manipulators. IEEE Transactions on Industrial Electronics, 39, 6, 565-569.

Kuan, C., and Hornik, K. (1991). Convergence of Learning Algorithms with Constant Learning Rates. IEEE Transactions on Neural Networks, 2, 5, 484-489.

Kuo, B. C. (1991). Automatic Control Systems. Prentice Hall, Englewood Cliffs, New Jersey, 475-477.

Kwon, T.M. and M.E. Zervakis, M.E. (1994). A Self-Organizing KNN Fuzzy Controller and its Neural Network Structure, International Journal of Adaptive Control and Signal Processing, Vol. 8, 407-431, 1994.

Masters, T. (1993). Practical Neural Network Recipes in C++. Academic Press, Inc., San Diego, 94-96, 245-249.

Miller, W. T. (1990). CMAC: An Associative Neural Network Alternative to Backpropagation. Proceedings of the IEEE, 78, 10, 233-239.

Miller, W. T. (1994). Real Time Neural Network Control of a Biped Walking Robot. IEEE Control Systems, 2, 41-48.

Miller, W. T., Glanz, F. H., and Kraft, L. G. (1987). Application of a General Learning Algorithm to the Control of Robotic Manipulators. The International Journal of Robotics Research, 6, 2, 84-98.

Miyamoto, H, Kawato, M., Setoyama, T., and Suzuki, R. (1988). Feed-Error-Learning Neural Network for Trajectory Control of a Robotic Manipulator. Neural Networks, 1, 251-265.

Morgan, J. S., Patterson, E. C., and Klopf, H. (1990). Drive-Reinforcement Learning: a Self-Supervised Model for Adaptive Control. Network, 1, 439-448.

Nagata, S., Sekiguchi, M., and Asakawa, K. (1990). Mobile Robot Control by a structured Hierarchical Neural Network. IEEE Control Systems Magazine, 4, 69-76.

Parks, P.C. and Militzer, J. (1991). Improved Allocation of Weights for Associative Memory Storage for Learning Control Systems, Proc. 1st IFAC Symp. on Design Methods for Control Systems, Zurich, Pergamon Press II, pp. 777-782.

Raibert, M. H. (1986). Legged Robots That Balance. MIT Press, Cambridge

Roberson, R. E., and Schwertassek, R. (1988). Dynamics of Multibody Systems. Springer-Verlag, Berlin

Rumelhart, D. E. and McClelland, J. L. (1986). Parallel Distributed Processing: Exploration in the Microstructure of Cognition, 1, MIT Press, Cambridge, MA.

Shelton, R. O., and Peterson, J. K. (1992). Controlling a Truck With an Adaptive Critic CMAC Design. Simulations, 5, 319-326.

Shih, C. and Gruver, W. A. (1992). Control of a Biped Robot in the Double-Support Phase. IEEE Transactions on Systems, Man, and Cybernetics, 22, 4, 729-734.

Sundararajan, N., Chin, L., and San, Y. K. (1993). Selection of Network and Learning Parameters for an Adaptive Neural Robotic Control Scheme. Mechatronics, 3, 6, 747-766.

Todd, D. J. (1985). Walking Machines, An Introduction To Legged Robots. Chapman and Hall, New York

Venugopal, K. P., Pandya, A. S., and Sudhakar, R. (1994). A Recurrent Neural Network Controller and Learning Algorithm for the On-Line Learning Control of Autonomous Underwater Vehicles. Neural Networks, 7, 5, 833-846.

Venugopal, K. P., Sudhakar, R., and Pandya, A. S. (1992). On-Line Learning Control of Autonomous Underwater Vehicles Using Feedforward Neural Networks. IEEE Journal of Oceanic Engineering, 17, 4, 308-318.

Wang, H., Lee, T. T. and Gruver, W. A. (1992). A neuromorphic controller for a three-link biped robot. IEEE Trans. on Sys., Man, and Cybern, 22, 1, 164-169.

Wang, J., and Malakooti , B. (1993). Characterization of Training Errors in Supervised Learning Using Gradient-Based Rules. Neural Networks, 6, 1073-1087.

Wang, L. and Mendel, J. M. (1992). Generating Fuzzy Rules by Learning from Examples. IEEE Trans.on Sys., Man and Cybern., 22, 6, 1414-1427.

Whitley, D., Mominic, S., Das, R., and Anderson, C. W. (1993). Genetic Reinforcement Learning for Neurocontrol Problems. Machine Learning, 13, 259-284.

Wu, Q. H., and Pugh, A. C. (1993). Reinforcement Learning Control of Unknown Dynamic Systems. IEE Proceedings-D, 140, 5, 313-322.

Zhang, W. (1995). Learning, tuning, and brainstorming: a multiagent cerebellar model for fuzzy-neural control of uniped robot locomotion (Part I and II). Working paper.

Zhang, W. (1996). NPN fuzzy sets and NPN qualitative algebra: a computational framework for bipolar cognitive modeling and multiagent decision analysis. IEEE Trans. Sys., Man, and Cybern. in press, 26, 8.

Zhang, W., Chen, S., Wang, W. and King, R. (1992). A CM-based approach to the coordination of distributed cooperative agents. IEEE Trans. on Sys., Man, and Cybern. 22, 1, 103-114.

Zheng, Y. F. (1989). Acceleration Compensation for Biped Robots to Reject External Disturbances. IEEE Transactions on Systems, Man, and Cybernetics, 19, 1, 74-84.

Zheng, Y. F. and Shen, J. (1990). Gait Synthesis for the SD-2 Biped Robot to Climb Sloping Surface. IEEE Transactions on Robotics and Automation, 6, 1, 86-96.