

# BIPED DYNAMIC WALKING USING REINFORCEMENT LEARNING

By

Hamid Benbrahim

DISSERTATION

Submitted to the University of New Hampshire in Partial Fulfillment of the  
Requirements for the Degree of

Doctor of Philosophy

in

Electrical Engineering

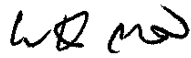
December, 1996

All Rights Reserved

c 1996

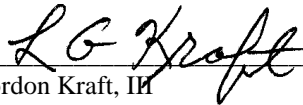
Hamid Benbrahim

This dissertation has been examined and approved.



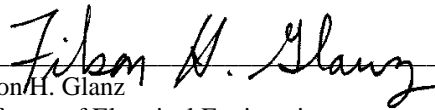
---

Dissertation Director, W. Thomas Miller III  
Professor of Electrical Engineering  
Chairman of the Department of Electrical and Computer Engineering



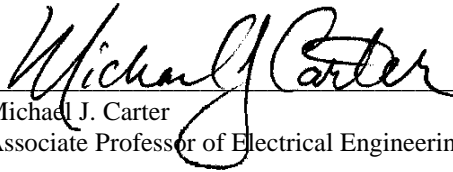
---

L. Gordon Kraft, III  
Professor of Electrical Engineering



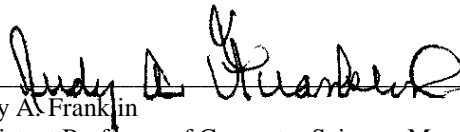
---

Filson H. Glanz  
Professor of Electrical Engineering



---

Michael J. Carter  
Associate Professor of Electrical Engineering



---

Judy A. Franklin  
Assistant Professor of Computer Science, Mount Holyoke College



---

Date

To Greta.

## ACKNOWLEDGMENTS

I would like to thank my advisor Dr. W. Thomas Miller. He has always been eager to help me in the multiple challenges that I faced during this study, from the most elementary problems to defining a direction for my research. He made me the envy of my colleagues by giving me full support while allowing me total freedom in conducting my research.

I would like to thank Dr. Judy A. Franklin who has always aggressively supported me during my research. She has been truly dedicated to this work and always generous with her rich scientific knowledge as well as her legendary human kindness.

I would like to thank Dr. L. Gordon Kraft, Dr. Filson H. Glanz and Dr. Michael J. Carter who have always believed in me, respected my research, and treated me like a friend. They have always been available to answer my numerous questions, and extremely generous about their valuable and constructive comments which greatly enhanced the quality of this work.

I would like to thank GTE Laboratories for funding this research and for giving me unconditional support. I would also like to thank the machine shop operator Jim Teal who did a great job making the biped's mechanical parts.

I would like to thank both my families Benbrahim and Meszoely for their moral and material support.

I would like to thank my wife Greta M. Meszoely. She has always used her tremendous liveliness and wisdom to give me the thrust that I desperately needed during this long tedious journey.

Finally I would like to apologize to all the people who helped me during this study, and who I did not have the chance to thank appropriately. I am forever grateful and will never forget to share what they have so generously given me.

# TABLE OF CONTENTS

<b>INTRODUCTION</b>	<b>12</b>
<b>BIPED WALKING</b>	<b>15</b>
<b>1 HISTORY</b>	<b>15</b>
<b>2 STATIC WALKING</b>	<b>16</b>
<b>3 DYNAMIC WALKING</b>	<b>17</b>
<b>4 INVERTED PENDULUM MODEL</b>	<b>19</b>
<b>5 CPG METHODS</b>	<b>20</b>
<b>6 QUICK OVERVIEW OF EXISTING BIPED ROBOTS</b>	<b>21</b>
<b>7 SUMMARY</b>	<b>24</b>
<b>LEARNING</b>	<b>25</b>
<b>1 SUPERVISED LEARNING</b>	<b>25</b>
<b>2 REINFORCEMENT LEARNING</b>	<b>26</b>
2.1 ACTOR-CRITIC	26
2.2 TD LEARNING	27
2.3 Q LEARNING	27
2.4 DISCRETE VS. CONTINUOUS ACTION DOMAIN	29
2.5 BANG-BANG CONTROL	30
2.6 THE STOCHASTIC REAL-VALUED (SRV) ALGORITHM	30
2.7 THE SELF SCALING REINFORCEMENT ALGORITHM (SSR)	32
2.8 ELIGIBILITY TRACES	35
<b>3 NEURAL NETWORKS</b>	<b>36</b>
3.1 BOXES	37
3.2 BACKPROPAGATION	38
3.3 CMAC	40
<b>4 OUTSTANDING CHALLENGES</b>	<b>42</b>
4.1 SENSOR ADAPTATION	43
4.2 LEARNING FROM ANALOGIES	43
<b>THE BIPED</b>	<b>44</b>
<b>1 MECHANICAL STRUCTURE</b>	<b>44</b>
<b>2 MECHANICAL SPECIFICATIONS</b>	<b>47</b>

<b>3</b>	<b>KINEMATICS</b>	<b>47</b>
<b>4</b>	<b>CONTROL STRUCTURE</b>	<b>48</b>
<b>5</b>	<b>COMMUNICATIONS INTERFACE</b>	<b>50</b>
<b><u>THE LEARNING ARCHITECTURE</u></b>		<b><u>51</u></b>
<b>1</b>	<b>GENERAL APPROACH</b>	<b>51</b>
1.1	MODULAR ARCHITECTURE	53
1.2	KNOWLEDGE INCORPORATION	53
1.3	TASK DEFINITION	54
<b>2</b>	<b>CENTRAL CONTROLLER</b>	<b>55</b>
<b>3</b>	<b>PERIPHERAL CONTROLLERS</b>	<b>56</b>
3.1	BODY POSTURE	56
3.2	BODY HEIGHT	57
3.3	STEP CONSTRAINT	58
3.4	TOTAL ENERGY	58
<b>4</b>	<b>REINFORCEMENT LEARNING</b>	<b>59</b>
4.1	THE ACTOR	59
4.2	THE CRITIC	61
<b><u>EXPERIMENTS &amp; RESULTS</u></b>		<b><u>64</u></b>
<b>1</b>	<b>IMPLEMENTATION ISSUES</b>	<b>64</b>
<b>2</b>	<b>IDENTIFICATION OF EFFECTIVE CONTROL VARIABLES</b>	<b>65</b>
2.1	ANKLE JOINT CONTROL	65
2.2	HIP JOINT CONTROL	65
2.3	KNEE JOINT CONTROL	66
<b>3</b>	<b>ACTION GENERATION</b>	<b>66</b>
3.1	BANG-BANG CONTROL	67
<b>4</b>	<b>CMAC PRE-TRAINING</b>	<b>68</b>
4.1	CPG CMAC PRE-TRAINING	68
4.2	BODY HEIGHT CMAC PRE-TRAINING	69
4.3	POSTURE CMAC PRE-TRAINING	70
4.4	CRITIC PRE-TRAINING	71
<b>5</b>	<b>THE SIMULATOR</b>	<b>71</b>
<b>6</b>	<b>THE BIPED</b>	<b>72</b>
6.1	STABILITY ISSUES	72
6.2	FALLING SAFEGUARD	74
6.3	EXTRA LEARNING WITH BANG-BANG CONTROL	74
<b>7</b>	<b>RESULTS</b>	<b>75</b>
7.1	SIMULATOR LEARNING CURVE	75
7.2	HARDWARE BIPED LEARNING CURVE	75
7.3	FOOT POSITIONS	76
7.4	REAL JOINT POSITIONS	78

7.5	CPG CMAC LEARNING	80
7.6	JOINT LEVEL CONTROL GRAPHS	82
7.7	BODY HEIGHT	84
7.8	BODY POSTURE	84
7.9	POSTURE INTERVENTION	85
7.10	FOOT PATTERNS ON THE FLOOR	85
7.11	BIPED MOVIE FRAMES	86
<b><u>CONCLUSION</u></b>		<b><u>88</u></b>
<b><u>REFERENCES</u></b>		<b><u>91</u></b>



## LIST OF FIGURES

Figure 1: Foot model	15
Figure 2: Leg model	16
Figure 3: Static walking	17
Figure 4: Dynamic walking	18
Figure 5: Forces applied to a link	18
Figure 6: Inverted pendulum	20
Figure 7: Distal learning	22
Figure 8: TD Learning in a continuous input domain	27
Figure 9: Q Learning in a continuous action domain	29
Figure 10: Self Scaling Reinforcement	33
Figure 11: SSR convergence	34
Figure 12: The Ball Balancer	35
Figure 13: Artificial neural network	36
Figure 14: Two layer neural network using boxes	37
Figure 15: Two layer backpropagation artificial neural network	38
Figure 16: CMAC neural network input/output representation	41
Figure 17: Biped views	44
Figure 18: The Biped	45
Figure 19: Joint structure	46
Figure 20: Mechanical specifications	47
Figure 21: Biped kinematics	48
Figure 22: Control structure	49
Figure 23: PID controller board	49
Figure 24: Architecture	52
Figure 25: Central controller	55
Figure 26: Body posture peripheral controller	56
Figure 27: Body height peripheral controller	57
Figure 28: Step constraint	58
Figure 29: The actor	60
Figure 30: General actor configuration	60
Figure 31: The CPG critic	62
Figure 32: Continuous action using bang-bang control	67
Figure 33: Pre-training foot positions	68
Figure 34: Posture CMAC pre-training	70
Figure 35: Step length command	71
Figure 36: Step response	73
Figure 37: CPG learning curve	75
Figure 38: Extra learning curve	76
Figure 39: Real foot positions: $x_r$ , $y_r$	77
Figure 40: Real foot positions: $x_l$ , $y_l$	77
Figure 41: Real foot positions: $x_r$ , $x_l$	78
Figure 42: Real foot positions: $y_r$ , $y_l$	78
Figure 43: Real joint positions: hips	79
Figure 44: Real joint positions: knees	79
Figure 45: Real joint positions: ankles	80
Figure 46: Ideal vs CPG joint positions: hip	81
Figure 47: Ideal vs CPG joint positions: knees	81
Figure 48: Ideal vs CPG joint positions: ankles	82

<i>Figure 49: Hip joint control</i>	82
<i>Figure 50: Knee joint control</i>	83
<i>Figure 51: Ankle joint control</i>	83
<i>Figure 52: Body height</i>	84
<i>Figure 53: Body height</i>	84
<i>Figure 54: Body posture</i>	85
<i>Figure 55: Posture constraint intervention frequency</i>	85
<i>Figure 56: Typical foot placements</i>	86
<i>Figure 57: Biped movie frames</i>	87

# ABSTRACT

## BIPED DYNAMIC WALKING USING REINFORCEMENT LEARNING

By

Hamid Benbrahim  
University of New Hampshire, December 1996

This thesis presents a study of biped dynamic walking using reinforcement learning. A hardware biped robot was built. It uses low gear ratio DC motors in order to provide free leg movements. The Self Scaling Reinforcement learning algorithm was developed in order to deal with the problem of reinforcement learning in continuous action domains. A new learning architecture was designed to solve complex control problems. It uses different modules that consist of simple controllers and small neural networks. The architecture allows for easy incorporation of modules that represent new knowledge, or new requirements for the desired task. Control experiments were carried out using a simulator and the physical biped. The biped learned dynamic walking on flat surfaces without any previous knowledge about its dynamic model.

# CHAPTER ONE

## INTRODUCTION

In the pursuit of solving complex control problems, many different approaches have been thoroughly investigated. Problem solving methods range from developing a complete model of the system and solving its equations to random trial and error. Each one of these methods can be successful in particular situations while being completely impractical in others. The success or failure of these methods depends on the type of nonlinearity of the system, nature of perturbations, change over time of the system characteristics or the environment, degree of instability, number and range of relevant inputs, etc. Many encountered systems do not fit in any specific category; they contain elements that require different problem solving methods in order to find a complete solution. It is thus necessary to find ways to integrate these methods and take advantage of each of their particular strengths.

Animals provide probably the best example of problem solving using different methods and taking advantage of all kinds of knowledge in order to perform a specific task. A human driving a car, for example, tries to stay at a constant distance from the next vehicle by using a PID control like method, accesses acquired knowledge about the state of the brakes to know how soon to use them, uses previous experience with the particular car to know how to handle sharp turns, etc. While doing all these control actions the driver is not necessarily conscious about them and does not always think about what method to use every time a new situation is encountered. This is an example of a control system that integrates several kinds of problem solving techniques to achieve a particular task.

Walking robots are becoming more and more interesting as computing power is getting more accessible and easily used for control. They have always been at the center of interest in robotics because of their ability to navigate in rugged terrain where wheeled robots cannot move. The research progress, however, has been relatively slow because of the complexity of legged robot dynamics. Walking robots are highly unstable, nonlinear, and depend on large numbers of parameters. So far most of the successful robots have four or more legs because they are inherently more stable than bipeds.

The reason biped locomotion is the center of this research is not because of a pure quest for complexity. Bipeds are extremely useful. They are the smallest of all the locomotion robots. Easy to carry around and requiring less energy, they can access areas that other robots cannot like ladders or very narrow paths, and they do not require change in the

working environment since they use the same kind of locomotion humans use. They can also be used as a prosthetic device to replace human legs or help handicapped people to walk.

As a control problem, biped robots are very challenging. They are inherently unstable because while lifting one leg the robot is standing on only one leg and the center of gravity is not always above the supporting foot. They are also highly nonlinear and the control system must be very smooth because any sharp movement can throw the robot hopelessly out of balance.

Despite all of these difficulties, humans seem to walk with no problem and without even noticing their control actions. This at least suggests that there is a very good solution to the problem. The fact that walking is most of the time done unconsciously suggests that maybe it does not require constant heavy computing in normal walking conditions. As in the car driving example humans successfully integrate different problem solving techniques in the walking process. This research relies heavily on these assumptions in trying to solve the biped dynamic walking problem.

Neural networks and learning methods prove to be very effective in controlling complex dynamic systems. They do not always require precise knowledge about the system's dynamics, and can learn solutions that cannot always be predicted by the user. They also continuously adapt to system and environment changes over time. Furthermore, it is obvious that animals acquire their skills mostly by learning and adaptation. It is thus necessary to explore these methods in depth, because of the great qualities they bring to control systems. It has been a thrust of this research to do so.

The research presented in this thesis started by applying different reinforcement learning algorithms to control a hardware two dimensional ball balancer. Both discrete and continuous action algorithms were investigated. A new continuous action reinforcement learning algorithm called Self Scaling Reinforcement (SSR) was invented as the result of these experiments. The particular details of the ball balancer experiments are not described in this document, while the resulting algorithms are thoroughly investigated.

A hardware biped robot was built to be the focus of this research, and to serve as the main testbed for the learning experiments. Since biped walking is more complex than ball balancing a new reinforcement learning architecture was designed for complex dynamic system control, and applied to the biped. This architecture succeeds in achieving dynamic biped walking without any knowledge of robot dynamics. The biped walks indefinitely on flat surfaces at slow speed (.5m/s).

This document consists of seven chapters. The introduction constitutes the first chapter. The second chapter presents a quick overview of existing biped walking research, and discusses the main issues related to biped dynamic walking. The third chapter introduces some relevant neural networks and learning methods. It also describes in detail the

reinforcement learning algorithms that are used in this research namely the SRV and SSR algorithms, as well as the CMAC neural network. The fourth chapter describes the hardware biped robot built as a part of this research. The fifth chapter presents the global learning architecture used to solve the problem of biped dynamic walking, also developed as part of this research. The sixth chapter describes the experiments and presents the results. The seventh, and last, chapter is the conclusion.

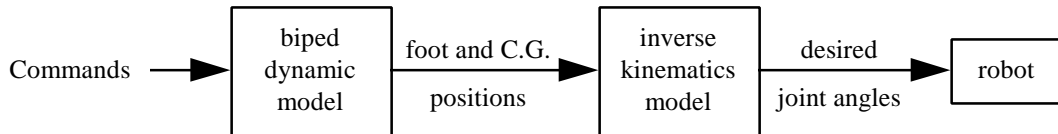
## CHAPTER TWO

### BIPED WALKING

#### 1 History

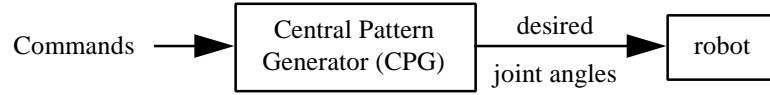
Although biped locomotion has been studied for a long time, it is only in the past twenty years, thanks to the fast development of computers, that real robots started to walk on two legs. Since then the problem has been tackled from different directions. First, there were robots that used static walking [Kato, 74]. The control architecture had to make sure that the projection of the center of gravity on the ground was always inside the foot support area. This approach was abandoned because only slow walking speeds could be achieved, and only on flat surfaces. Then, dynamic walking robots appeared [Takanishi, 82]. The center of gravity can be outside of the support area, but the zero momentum point (ZMP), which is the point where the total angular momentum is zero, cannot. Dynamic walkers can achieve faster walking speeds, running [Raibert, 84], stair climbing [Takanishi, 90], [Kurematsu, 91], execution of successive flips [Hodgins, 90], and even walking with no actuators [McGeer, 90]. Static and dynamic walkers are described in more detail in the next two sections.

The first dynamic walkers used the following architecture. First, the user supplies a command that represents the desired walking pattern. Then, a dynamic model of the biped, which is usually a simplified inverted pendulum model, determines the foot and center of gravity positions. The desired joint angles are obtained using the biped's inverse kinematics model, and a linear feedback controller makes the robot joints follow the desired trajectories (Figure 1). The model used here is referred to as foot model because it uses foot positions as control action.



*Figure 1: Foot model*

Later, another architecture that uses a Central Pattern Generator (CPG) appeared [Bay, 87]. The biped model and the inverse kinematics model in the previous architecture are replaced by a CPG that generates periodic signals that represent the desired joint trajectories (Figure 2). The model used here is referred to as leg model because it focuses on leg movements.



*Figure 2: Leg model*

These descriptions show only the general structure, while the actual implementations involve more complex control methods. These methods are examined later in this chapter.

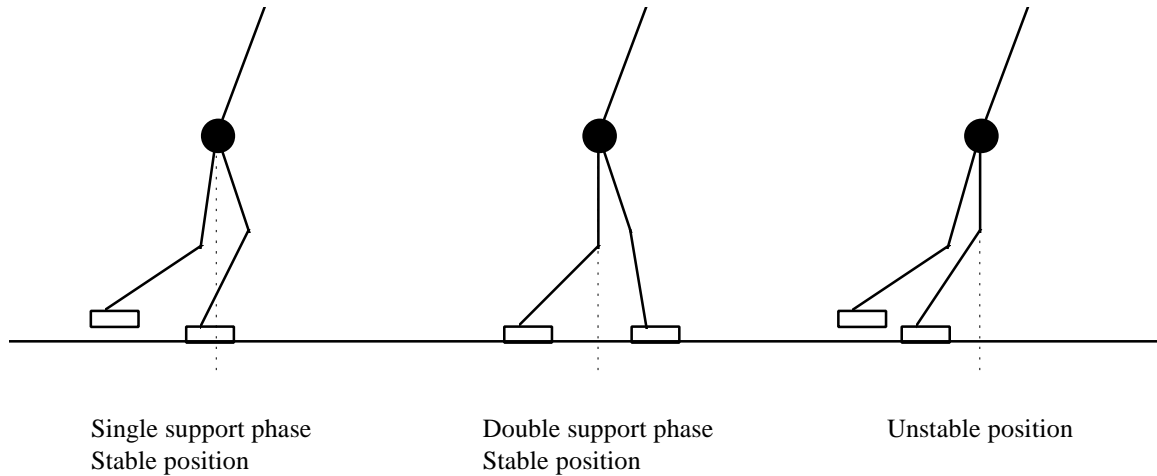
Another major step in biped locomotion was the introduction of neural networks and learning [Miller, 90, 94],[Zheng, 90]. With these methods, it is not necessary to know the robot's exact dynamic model, the walker can find solutions that the user cannot think of in advance, and it can adapt to environment changes. Miller and Zheng's work is described at the end of this chapter. It provides a starting point for this research.

## **2 Static walking**

Static walking assumes that the robot is statically stable. This means that, at any time, if all motion is stopped the robot will stay indefinitely in a stable position. It is necessary that the projection of the center of gravity of the robot on the ground must be contained within the foot support area (Figure 3). The support area is either the foot surface in case of one supporting leg or the minimum convex area containing both foot surfaces in case both feet are on the ground. These are referred to as single and double support phases, respectively. Also, walking speed must be low so that inertial forces are negligible.

This kind of walking requires large feet, strong ankle joints and can achieve only slow walking speeds. It has been abandoned by most researchers for dynamic walking, which provides more realistic and agile movements.



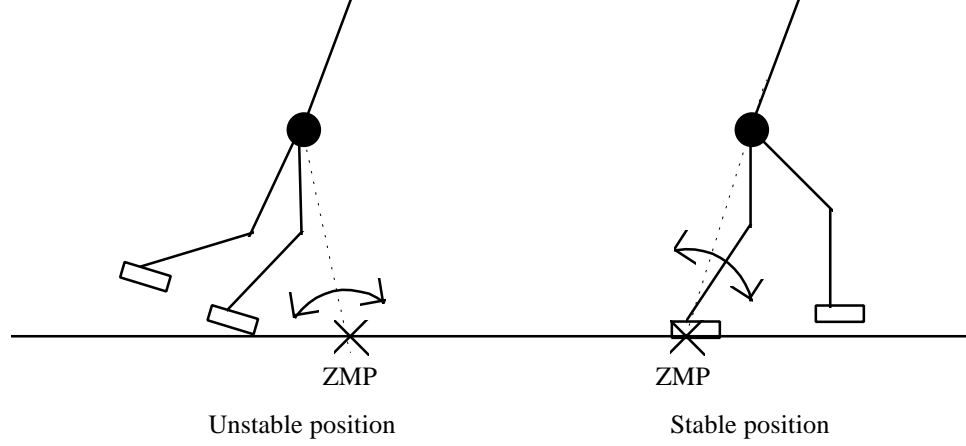


*Figure 3: Static walking*

### 3 Dynamic walking

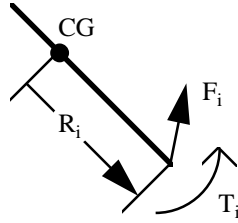
Biped dynamic walking allows the center of gravity to be outside the support region for limited amounts of time. There is no absolute criterion that determines whether the dynamic walking is stable or not. Indeed a walker can be designed to recover from different kinds of instabilities [Hodgins, 90],[Raibert, 84]. However, if the robot has active ankle joints and always keeps at least one foot flat on the ground then the Zero Momentum Point (ZMP) can be used as a stability criterion. The ZMP is the point where the robot's total moment at the ground is zero. As long as the ZMP is inside the support region the walking is considered dynamically stable because it is the only case where the foot can control the robot's posture. It is clear that for robots that do not continuously keep at least one foot on the ground or that do not have active ankle joints (walking on stilts), the notion of support area does not exist, therefore the ZMP criterion cannot be applied.

Dynamic walking is achieved by ensuring that the robot is always rotating around a point in the support region (Figure 4). If the robot rotates around a point outside the support region then this means that the supporting foot will tend to get off the ground or get pressed against the ground. Both cases lead to instability. To draw an analogy with static walking, if all motion is stopped then the robot will tend to rotate around the ZMP.



*Figure 4: Dynamic walking*

The position of the ZMP is computed by finding the point  $(X,Y,Z)$  where the total torque is zero. Since we are only interested in the ground plane we assume that  $Z=0$ . To avoid confusion, torque and moment mean the same thing here. The robot has  $n$  links; each link is subject to a total force  $F_i$  applied at a point determined by the vector  $R_i$  relative to the center of gravity of the link.  $T_i$  determines the total motor torque applied to the link.  $R_z$  is the ZMP vector and  $T$  is the robot's total torque. An example of the forces applied to a link is represented in Figure 5.



*Figure 5: Forces applied to a link*

The force, torque and position vectors have the following coordinates:

$$F_i: (F_{xi}, F_{yi}, F_{zi})$$

$$T_i: (T_{xi}, T_{yi}, T_{zi})$$

$$R_i: (x_i, y_i, z_i)$$

$$R_z: (X, Y, Z)$$

Then the total torque is computed as:

$$T = \sum_{i=1}^n (R_i + R_z) \times F_i + \sum_{i=1}^n T_i = 0 \quad (1)$$

where  $\times$  represents the cross product. Equation (1) is then expanded as:

$$\begin{aligned} \sum_{i=1}^n (y_i + Y) F_{zi} - \sum_{i=1}^n (z_i + Z) F_{yi} + \sum_{i=1}^n T_{xi} &= 0 \\ \sum_{i=1}^n (z_i + Z) F_{xi} - \sum_{i=1}^n (x_i + X) F_{zi} + \sum_{i=1}^n T_{yi} &= 0 \\ \sum_{i=1}^n (x_i + X) F_{yi} - \sum_{i=1}^n (y_i + Y) F_{xi} + \sum_{i=1}^n T_{zi} &= 0 \end{aligned} \quad (2)$$

Making  $Z=0$  and solving these equations for  $X$  and  $Y$  we obtain the ZMP coordinates:

$$\begin{aligned} X &= \frac{\sum_{i=1}^n (z_i F_{xi} - x_i F_{zi}) + \sum_{i=1}^n T_{yi}}{\sum_{i=1}^n F_{zi}} \\ Y &= \frac{\sum_{i=1}^n (z_i F_{yi} - y_i F_{zi}) + \sum_{i=1}^n T_{xi}}{\sum_{i=1}^n F_{zi}} \end{aligned} \quad (3)$$

## 4 Inverted pendulum model

The inverted pendulum is the most commonly used dynamic model to represent the biped's dynamics. This is only valid if there always is one and only one foot on the ground. The robot is represented by a point mass equal to the total mass of the robot and situated at the center of gravity, and a massless beam pivoting around the point of foot contact with the ground. Notice that for an inverted pendulum, the ZMP is located at the point of contact with the ground.

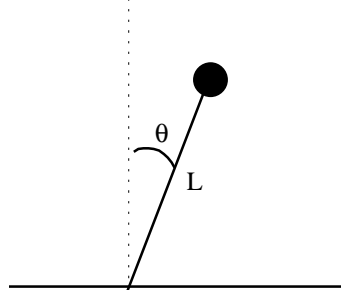


Figure 6: Inverted pendulum

The dynamic equation of the inverted pendulum is:

$$x(\ddot{y} + g) - y\ddot{x} = 0 \quad (4)$$

Where  $x$  and  $y$  are the coordinates of the center of gravity,  $L$  is the length of the pendulum,  $\theta$  is the pendulum's deviation from the vertical and  $g$  is the gravity. This is a highly nonlinear equation, it can be simplified by assuming constant height as follows:

$$\ddot{x} = \left(\frac{g}{h}\right)x \quad (5)$$

where  $h$  is the constant body height. This is a commonly used assumption since most humans and walking robots keep the body at constant height. This provides large savings on energy because since the body is the heaviest part of the biped, body movement can dramatically increase kinetic energy expenditure. The length of the pendulum changes according to the robot's knee joint, or whatever mechanism is used to replace the knee function.

This is an insufficient model since it does not take into consideration the energy related to changing the pendulum's length. It is however a reasonable approximation at slow walking speed.

## 5 CPG methods

Central Pattern Generators (CPG) have been identified in most animals as being responsible for generating periodic signals that excite motion muscles [Grillner, 76]. They can be mathematically formed by a set of coupled oscillators, some of which are excitatory and others inhibitory. When configured properly these oscillators generate the

complete walking pattern. They can also generate different types of walking, i.e. jumping, crawling, running etc.

Van Der Pol [Pol, 26] generated a set of equations that are now widely used as CPG oscillators. The oscillators are defined as follows:

$$\begin{aligned}\ddot{x}_i - \mu_i (p_i^2 - c_i^2) \dot{x}_i + g_i^2 c_i &= q_i \\ c_i &= x_i - \sum_{j \neq i} \lambda_{ji} x_j\end{aligned}\tag{6}$$

where  $x_i$  are the outputs of the oscillators,  $p_i^2$  are called amplitude parameters,  $q_i$  offset parameters,  $g_i^2$  frequency parameters and  $\lambda_{ij}$  are called coupling parameters.

## 6 Quick overview of existing biped robots

This section presents a brief description of some biped locomotion realizations.

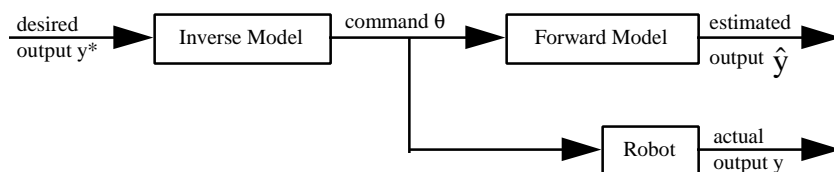
Miller's biped [Miller, 94] uses a basic Central Pattern Generator (CPG) to generate desired joint positions according to user supplied commands. This CPG consists of a set of controllers. The CPG's output is determined by a set of parameters that a CMAC neural network (discussed in Chapter 3) learns in order to achieve stable dynamic walking. By monitoring the differences between left and right, and front and back foot forces, the CMAC can determine whether the walking is stable or not. The CPG's user supplied input commands represent the desired walking pattern: step length, step interval, and lift magnitude. Another CMAC learns kinematically consistent postures to ensure reasonable joint position commands. The biped can successfully walk on flat surfaces relying only on very simple dynamics and kinematics models.

Bay and Hemami [Bay, 87] use coupled Van Der Pol oscillators as a CPG. The parameters of these oscillators govern the magnitude, frequency, and offset of each oscillator. Additional parameters are used to couple the individual oscillators together to make them excite or inhibit each other, and work in a synchronized manner. By choosing the appropriate set of parameter values (mostly using trial and error), the oscillators generate signals that are used as desired joint trajectories for a biped walking robot. Different sets of parameter values can achieve different walking patterns.

Zheng [Zheng, 90] uses a knowledge base where a few CPG trajectories are stored. The biped detects the floor's slope using its foot sensors, and looks in the knowledge base for a corresponding pattern. If one exists it uses it, otherwise, it chooses the closest pattern and uses a neural network to modify it to suit the new terrain. Once the new pattern has

been learned, it is stored in the knowledge base. With this method the robot learns to walk on flat and variably slopping surfaces. Zheng distinguishes between two types of behavior. When the walker finds a familiar terrain it uses an existing pattern, and does not need feedback. This is called voluntary motion. If the terrain, however, is not familiar the walker needs feedback in order to adjust its joint positions. This is called reflexive motion. Wagner et al. [Wagner, 88] use a slightly different approach. They find a set of optimal solutions through simulation and store them in a knowledge base. These solutions are invoked during walking and modified according to sensory inputs.

Stitt and Zheng [Stitt, 93] apply distal learning [Jordan 92] to control a biped. This method incorporates a forward model of the robot dynamics and uses it to convert stability information into information on how to adjust the robot's joints so as to regain stability. In distal supervised learning (Figure 7) the correct joint position is unknown. However, target values for the outcome of adjusted joint positions are known (i.e. foot forces). If the gait is adequate, the feet will provide a stable support. Thus a desired foot force trajectory is known. This is called a distal variable. When the robot walks using the wrong gait, the unbalanced condition will be sensed by foot force sensors. A neural network is trained to learn the relationship between the joint adjustments and the foot forces. Miller et al. [Miller 90] use a similar approach to control a robot arm. A CMAC neural network learns the inverse model of the robot, and generates the appropriate command to track a desired trajectory.



*Figure 7: Distal learning*

Kajita et al. [Kajita, 92] use the inverted pendulum to model the robot dynamics. By assuming constant body height and posture, they obtain a simple differential equation representing the position of the center of gravity. They also obtain an energy like expression that is constant for each stable walking pattern. By defining the successive values of these energies and the appropriate step lengths, the robot can achieve stable walking and change patterns. This method assumes that the robot's model is very close to an inverted pendulum.

Wang et al. [Wang, 92] use a modular neural network strategy. They classify the walking cycle into five phases, and decouple the dynamics equations in order to get independent joint equations. They implement the controller as follows: each joint is controlled by a

separate supnet, and each supnet consists of five subnets corresponding to the five phases of walking. All the networks are integrated into a global net. They use backpropagation and supervised learning, based on their dynamics model, to update the parameters of these networks. Katic and Vukobratovic [Katic, 92] use a similar method. They decompose the dynamics of the robot into several equations and train a network for each equation. They increase convergence speed in the neural networks by considering the problem of adjusting the weights as a problem of estimating parameters. They use the recursive LS method and the Extended Kalman Filter.

Takanishi et al. [Takanishi, 90] introduce a new concept, called “Virtual Surface”, to consider the ZMP (Zero Moment Point) on an uneven surface. The robot can climb stairs and walk on inclined terrain. They control the trunk’s position to deal with the problem of stability and the lower limbs to do the path control of dynamic walking independently. The virtual surface is represented by the polygon formed by the 2 or 4 support points (heels and toes) whether the robot is in single or double support phase. The trajectory of the ZMP is then planned within this surface. The lower limbs move according to a predefined trajectory and the trunk controls the ZMP to follow the desired trajectory within the virtual surface. They linearize the ZMP equations in order to compute the appropriate control signals, and use Fast Fourier Transform (FFT) to find periodic solutions.

Kurematsu et al. [Kurematsu, 91] use the inverted pendulum model to generate joint trajectories. A neural network learns to compensate for the difference between the biped and the model, and another neural network learns the inverse kinematics. The robot learns to walk on stairs and flat surfaces. A similar approach is used by Kitamura et al. [Kitamura, 90].

Raibert [Raibert, 84] built a series of hopping robots. They have telescopic legs with no knees and no ankles. The legs contain springs that make the robot bounce when it falls on its feet. By changing the length of the springs, the controller adjusts the energy stored in the robot, thus controls the hopping height and the running speed. The leg angles are chosen according to the desired running speed or other desired behavior like executing forward flips [Hodgins, 90].

Shih et al. [Shih 91, 92] optimize the walking by minimizing the difference between the Zero Momentum Point (ZMP), which is the virtual total ground reaction point, and the center of the supporting area. They use a dynamics model and a kinematics model to generate the appropriate joint trajectories.

Goddard et al. [Goddard, 92] control their biped by determining the appropriate toe push-off forces. They derive the dynamics equations of the biped, determine the nominal motion according to the sensed terrain slope, and find the appropriate toe push-off force. They use feedforward gait generation and feedback stabilization.

Sano et al. [Sano, 90] use the inverted pendulum model and control the angular momentum at the ankle.

Miura and Shimoyama [Miura, 84] built five bipeds BIPER 1, 2, 3, 4, 5. The robots are not statically stable because they have no ankle joint actuators. They linearize the inverted pendulum model and generate the joint trajectories that satisfy the model's equations, then they use linear feedback to control joint positions.

Kawaji et al. [Kawaji, 92] use an inverted pendulum model to generate a trajectory defined by the foot position of the free leg, and the position of the waist. They use inverse kinematics to determine the desired joint angles.

## **7 Summary**

The previous section shows a number of methods that have been used to solve the problem of biped locomotion. It is difficult to detect a specific trend, because most of these methods are being used and investigated at the same time. We can however detect some major breakthroughs that will definitely set new directions [Miller, 94],[Zheng, 90],[Bay, 87],[Wang, 92],[Stitt, 93].

The introduction of learning and neural networks to biped locomotion has shown better results than conventional control methods. Indeed, it is difficult to accurately model the dynamics of a biped and to find analytical control rules that will solve stability and nonlinearity problems. Furthermore, if one agrees with the concept that trying to mimic animal behavior is the most promising direction in robotics, then learning and adaptation, central pattern generation, pattern recognition, and modularity are the directions to take.

Miller introduces a method that uses a modular structure and a reflexive behavior, Zheng uses the concepts of voluntary and reflexive motion, and Bay and Hemami use a CPG that generates periodic joint angle trajectories. All these methods are very elegant and show great improvements over conventional methods.

Learning and neural networks methods have a great potential for solving the problem of biped locomotion. They have been tested in various linear and nonlinear control problems, and have shown great adaptability, immunity to noise, and robustness. Two major obstacles that these methods must overcome are the need for a large memory space and for computing power. These obstacles are quickly disappearing as fast progress is being achieved in computer technology.



# CHAPTER THREE

## LEARNING

This chapter describes some of the learning methods that are closely related to this research. The methods that are used in the biped learning as well as those that have been developed during this research are described in great depth, while the other methods are presented briefly.

The first section presents a brief definition of supervised learning. The second section presents reinforcement learning. It describes two methods that deal with the credit assignment problem, Temporal Differences (TD) learning and Q learning. The problem of continuous vs. discrete action domain in reinforcement learning is also presented, as well as different methods that try to solve it: Bang-bang, Stochastic Real Valued (SRV), Self Scaling Reinforcement (SSR). Eligibility traces are described under reinforcement learning even though they can be used in supervised learning too. The third section describes three neural network configurations: Boxes, Backpropagation, and CMAC. The fourth section discusses a few outstanding learning challenges.

### 1 Supervised learning

Supervised learning is based on the theory of adaptive control. It is an adaptation mechanism where the system learns to perform its designated task with the assistance of a teacher. When the system outputs an action, the teacher generates an error signal equal to the difference between the desired and actual actions. The system then updates its parameters in order to minimize the error. This of course requires that a desired action is known and the system is explicit enough to know how to update its parameters in a direction that minimizes the error.

When a system is composed of several cascaded modules the desired outputs of each module are not all known. It is possible to backpropagate the desired system output through all the modules by computing the gradient of the error, when analytically available [Rumelhart, 86], or by identifying the inverse of the modules for which a desired output is known [Jordan, 92].

## 2 Reinforcement learning

Reinforcement learning is used when very little knowledge is available about the system to be controlled. It is based on the following idea. The controller is assigned a specific task. If it succeeds in accomplishing it, it receives a reward (or a positive reinforcement), and if it fails it receives a punishment (or a negative reinforcement). The controller then learns, through experience, to avoid actions that yield punishment and to adopt actions that lead to success. It is closely related to the theory and methods of dynamic programming [Sato, 88], [Barto, 90] and optimal control, and has roots in the psychological study of classical conditioning [Barto, 83], [Sutton, 84].

In many situations the success or failure of the controller is determined not only by one action but by a succession of actions. The learning algorithm must thus reward each action accordingly. This is referred to as the problem of delayed reward. There are two basic methods that are very successful in solving this problem, TD learning, [Sutton, 84] and Q learning, [Watkins, 89]. Both methods build a state space value function that determines how close each state is to success or failure. Whenever the controller outputs an action, the system moves from one state to another. The controller parameters are then updated in the direction that increases the state value function.

When the action space is discrete, the implementation of reinforcement learning is straightforward, [Barto, 83], [Franklin, 88], [Benbrahim, 92]. When the system has to pick an action from a fixed set, it chooses the one that has obtained success more often than any of the others. When the action domain is continuous, the problem is less obvious. Statistical gradient following methods, [Williams, 92], [Gullapalli, 90], [Benbrahim, 94] have produced some promising results. These methods use a random number generator with a Gaussian distribution to generate the action. A neural network controls the mean and standard deviation of the Gaussian. The network weights are updated toward the direction that yields higher reinforcement. As a result, the mean converges toward an optimal action and the standard deviation increases when the system needs to search the action space, and converges toward zero once an optimal action policy has been learned.

### 2.1 Actor-Critic

In the Actor-Critic learning configuration there are two systems, an actor and a critic. The actor generates an action and the critic rewards the actor according to the outcome of that action. The critic builds a state evaluation function that predicts the final outcome of the system behavior, whether it will be success or failure. Some of the methods that are generally used to build these predictions in the Actor-Critic configuration are Temporal Differences (TD) learning and Q learning. These methods are described below.

## 2.2 TD learning

Temporal Differences (TD) learning [Barto, 83], [Sutton, 84] is used to solve the problem of delayed rewards in reinforcement learning. It can also be used in a variety of prediction tasks [Miller, 94]. Within an Actor-Critic configuration the critic uses TD learning and the raw reinforcement  $r$  to create a more developed reinforcement signal  $\hat{r}$  as follows:

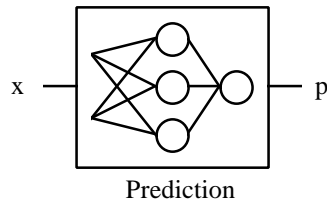
$$\hat{r} = r + \gamma p(x_k, t) - p(x_{k-1}, t-1) \quad (7)$$

where  $p(x)$  is the prediction of future success when the state is  $x$  and  $0 < \gamma < 1$  is a decay factor. If the system moves from one state  $x_{k-1}$  to another state  $x_k$  that has a higher prediction of success, then the action responsible for this move will be rewarded accordingly. The prediction is updated as follows:

$$p(x_k, t) = p(x_k, t_1) + \beta \hat{r} \quad (8)$$

where  $\beta$  is a positive decay factor  $< 1$  and  $t_1$  is the time where state  $x_k$  was last visited.

TD learning can be implemented for a continuous input domain by using a neural network that learns the predictions as shown in Figure 8. The network can be updated using LMS and using  $\beta \hat{r}$  as the LMS error. Neural networks and LMS are described in more detail in Section 3.



*Figure 8: TD Learning in a continuous input domain*

## 2.3 Q learning

Q learning [Watkins, 89] deals with the problem of delayed rewards by building a state-action evaluation function  $Q(x,a)$ , where  $x$  represents the state and  $a$  the action. TD learning [Sutton, 84] also builds an evaluation function  $V(x)$ , which does not represent

the actions explicitly as in Q learning. A very simplistic comparison between the two methods would be that  $V(x)$  could be considered as a rough approximation of the average of  $Q(x,a)$  for all possible actions.

If at time  $t$  the system moves from state  $x_k$  to  $x_{k+1}$  by executing action  $a$ , and receives a reinforcement  $r$ , then the  $Q$  values are updated as follows:

$$Q(x_k, a, t + 1) = \lambda Q(x_k, a, t) + (1 - \lambda)[r + \gamma \max_{b \in A} \{Q(x_{k+1}, b, t)\}] \quad (9)$$

where  $A$  is the set of all possible actions,  $\lambda$  is an averaging factor (between 0 and 1), and  $\gamma$  is a time discount factor that determines how far back, in time, the actions should be rewarded for the current outcome.

When the  $Q$  values converge to their optimal value function  $Q^*$ , then if the controller always chooses actions that correspond to maximum  $Q$  values, the system will reach an optimal goal.

Q learning is much more powerful than TD learning because it offers a more explicit state-action evaluation function. It requires, however, more computing power, and memory space, and finding the optimal policy is difficult in the general case. It can be successfully used as a replacement to more costly dynamic programming methods [Barto, 90].

The implementation of Q learning in the case of a discrete action domain is straight forward. The learning algorithm can scan the action space easily and search for the action that has the largest  $Q$  value. In a continuous action domain, however, scanning the action space is more difficult; this makes Q learning widely used only in discrete action domains.

One way to implement Q learning in a continuous action domain is to use two neural networks, one that learns the  $Q$  values corresponding to the input state  $x$  and action  $a$ , and another that learns the maximum value  $Q_{\max}$  of  $Q$  and the associated action  $a^*$  for the input state  $x$  (Figure 9). The action  $a$  is in general equal to  $a^*$ , plus a random component. This randomness is used to scan the action domain.

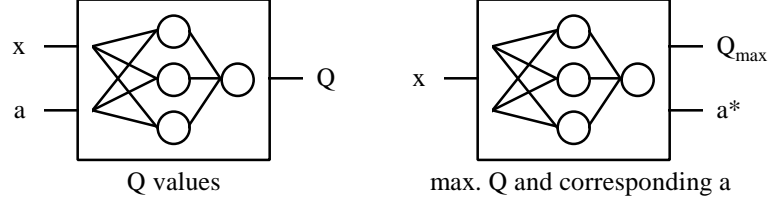


Figure 9:  $Q$  Learning in a continuous action domain

The two networks can be updated using LMS with the error signals  $\delta_1$  and  $\delta_2$ , respectively.

$$\mathbf{d} = (1 - I)(r + \mathcal{Q}_{\max} - Q(x_k, a^*, t)) \quad (10)$$

$$\mathbf{d} = \begin{bmatrix} \max\{Q(t+1), \mathcal{Q}_{\max}(t)\} - \mathcal{Q}_{\max}(t) \\ a(t+1) - a^*(t) \text{ if } Q(t+1) > \mathcal{Q}_{\max}(t), 0 \text{ otherwise} \end{bmatrix}$$

$Q$  learning is probably the most sophisticated reinforcement learning method. However, as mentioned earlier, because of its relative complexity, it is more realistic to apply it in discrete than in continuous action domains, especially in real-time control. It is possible to further optimize it in order to alleviate the computational cost and make it suitable for real-time control in continuous action domains.

## 2.4 Discrete vs. Continuous action domain

Although most of the learning theory in discrete and continuous action domains is similar, there are crucial differences in practice. Consider a stochastic learning system where the action is chosen randomly according to a certain probability distribution  $p()$ . The learning approach is to find a probability distribution that maximizes the likelihood of success. The probability that a certain action  $x$  will be chosen is equal to  $p(x)$ . If the generated action yields success then its probability will be increased to increase the likelihood that the action will occur [Narendra, 74].

In the case of a discrete action domain the action is chosen as follows. Each action is assigned an output value

$$v(x) = p(x) + r(x) \quad (11)$$

where  $r(x)$  is a random number. The random numbers  $r(x)$  are independent and have the same distribution. The learning algorithm scans the action domain and chooses the action that has the highest output value. It is not realistic to apply this method to continuous action domains. It takes a long time to scan the domain and compute the output value for each individual action.

Instead of trying to learn a probability distribution for a continuous action domain, a constant distribution is used, usually a Gaussian. The learning algorithm learns the appropriate mean and standard deviation for each input. To make an analogy with the discrete action case, the probability of a continuous action is increased by moving the mean of the Gaussian closer to it. The standard deviation usually decreases as the system learns.

The larger the reinforcement a discrete action gets the more likely it will be chosen in the future. We cannot say the same thing about the continuous action case. If the mean of the Gaussian is moved toward the action proportionally to the reinforcement then it might overshoot if the reinforcement is too large.

The Stochastic Real-valued (SRV) [Gullapalli-90] and Self Scaling Reinforcement (SSR) [Benbrahim, 94] algorithms deal with this problem in more detail.

## **2.5 Bang-bang control**

Bang-bang control is an old control method that uses 2 discrete actions. The controller usually chooses between a negative and a positive action of equal magnitude. If these magnitudes are chosen appropriately then it can be possible to control many systems.

This method is usually appropriate when the system dynamics respond favorably to sharp changes in the action and where accuracy is not of major concern. A system that can easily oscillate will be very hard to control with a bang-bang controller.

Since it uses discrete actions, a controller that is learning which action to take can learn very quickly and the controller can be relatively simple. This makes bang-bang a very attractive method in many control applications. It has been tried with success in learning to balance an inverted pendulum in simulation, [Barto, 83] and a real-time ball balancer [Benbrahim, 92] implemented in hardware.

## **2.6 The Stochastic Real-valued (SRV) Algorithm**

For completeness, we start by examining Williams' [Williams, 92] statistical gradient following algorithm that uses a stochastic output unit. This unit is a Gaussian random number generator with mean  $\mu$  and standard deviation  $\sigma$ . For every input vector  $x$ , the

algorithm generates  $\mu(x)$  and  $\sigma(x)$ . These are used to generate the action  $y(x)$  according to the probability distribution:

$$g(y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-\mu)^2}{2\sigma^2}} \quad (12)$$

where the input vector  $x$  is omitted for clarity.

The learning algorithm is governed by a weight vector  $w$ , that is updated by:

$$w(t+1) = w(t) + \alpha(r - \bar{r})e(w) \quad (13)$$

where  $0 < \alpha < 1$  is a learning rate,  $r$  is the reinforcement and  $\bar{r}$  is a reinforcement base line, a prediction of  $r$ . The characteristic eligibility  $e(w)$  is a measure of how eligible each weight is for updating. Williams defines the characteristic eligibility  $e(w)$  by the equation:

$$e(w) = \frac{\partial \ln g}{\partial w} \quad (14)$$

To better understand how this algorithm works consider these equations for the case where linear units produce  $\mu$  and  $\sigma$ :

$$\mu(x) = w_\mu^T x, \text{ and } \sigma(x) = w_\sigma^T x \quad (15)$$

$$e(w_\mu) = \frac{\partial \ln g}{\partial \mu} \frac{\partial \mu}{\partial w_\mu} = \frac{y - \mu}{\sigma^2} x \quad (16)$$

$$e(w_\sigma) = \frac{\partial \ln g}{\partial \sigma} \frac{\partial \sigma}{\partial w_\sigma} = \frac{(y - \mu)^2}{\sigma^3} x \quad (17)$$

The weight vectors are updated as follows:

$$w_\mu(t+1) = w_\mu(t) + \alpha(r - \bar{r}) \frac{y - \mu}{\sigma^2} x \quad (18)$$

$$w_{\sigma}(t+1) = w_{\sigma}(t) + \alpha(r - \bar{r}) \frac{(y - \mu)^2}{\sigma^3} x \quad (19)$$

Note that if the output yields a reinforcement better than predicted,  $\mu$  will be updated to move toward  $y$ . This will increase the probability that  $y$  is the action to be chosen next time the same input  $x$  occurs. And if  $r$  is less than predicted,  $\mu$  will move away from  $y$  to decrease this probability. The standard deviation  $\sigma$  increases or decreases to determine the extent of action domain exploration. Williams proves the convergence of these types of algorithms.

Even though these methods converge, we found that in practical applications, the learning is very slow and it is very hard to find appropriate learning rates. Notice also that when  $\sigma$  is very small, the term  $1/\sigma^3$  overflows.

Gullapalli's SRV algorithm presents a significant improvement [Gullapalli, 90, 94]. It updates the unit's parameters as follows:

$$w_{\mu}(t+1) = w_{\mu}(t) + \alpha(r - \bar{r}) \frac{y - \mu}{\sigma} x \quad (20)$$

$$\sigma = k(\max(r) - \bar{r}) \quad (21)$$

where  $k$  is a multiplying factor. When the system has learned, the predicted reinforcement becomes close to the maximum reinforcement and  $\sigma$  becomes very small. This means that the action will be essentially equal to  $\mu$ , and thus the search will be stopped.

## 2.7 The Self Scaling Reinforcement algorithm (SSR)

Williams' and Gullapalli's methods suffer from a major handicap: if the system gets a very large reinforcement, let us say infinite, the weights will overshoot and therefore impede the system performance. If the reinforcement range of variation is known, the learning rate can then be adjusted to prevent overshooting. When this is not the case, however, a very small learning rate must be used, and a slow learning process ensues.

SSR [Benbrahim, 94] is a new reinforcement learning algorithm that was developed during this research. It is based on the concept that if a certain action yields an infinite reinforcement then it should have its probability greatly increased. Moreover, since it is not guaranteed that the system will receive the same level of reinforcement for different



input vectors, the notion of infinite reinforcement should be flexible. The best reinforcement the system gets is considered infinite and the worst is considered negatively infinite. The algorithm keeps track of the maximum and minimum reinforcement,  $r_{\max}$  and  $r_{\min}$  respectively, and computes a scaled reinforcement as follows:

$$\hat{r} = \exp(r - r_{\max}) - \exp(r_{\min} - r) \quad (22)$$

Notice that if  $r = r_{\max}$ , then,  $\hat{r} = 1 - \exp(r_{\min} - r_{\max}) = 1 - \epsilon$  and if  $r = r_{\min}$ , then  $\hat{r} = \exp(r_{\min} - r_{\max}) - 1 = \epsilon - 1$ ,  $\epsilon$  is very small when  $r_{\min} - r_{\max} \ll 0$ ; consequently, it can be ignored for clarity purposes.

Figure 10 shows three graphs of the SSR as a function of  $r$  for large, medium and small  $r_{\max} - r_{\min}$ , where  $\epsilon$  is not ignored. Notice that  $\epsilon$  increases as  $r_{\max} - r_{\min}$  decreases. For large values of  $r_{\max} - r_{\min}$ , only extreme reinforcement is taken into account. This means that the system uses very low resolution for the reinforcement signal in the beginning. As  $r_{\max} - r_{\min}$  diminishes the SSR function becomes more linear, thus a finer resolution is used. As  $r_{\max} - r_{\min}$  converges toward zero, the SSR function converges toward the  $x$  axis. The learning then stops. If a new reinforcement occurs outside the  $[r_{\min}, r_{\max}]$  segment, then  $r_{\max} - r_{\min}$  automatically increases and more learning ensues.

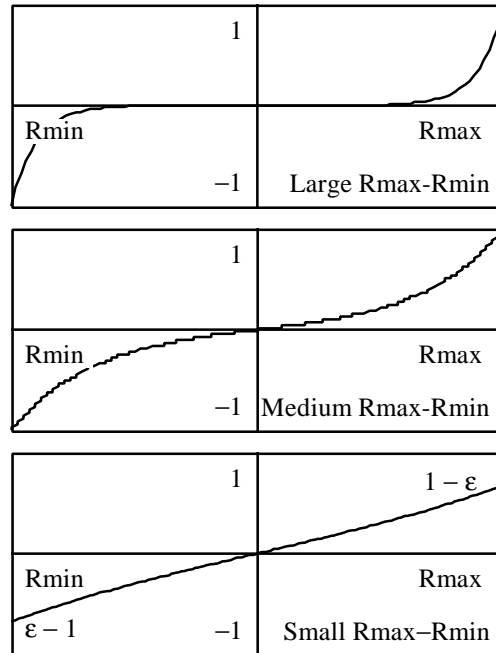


Figure 10: Self Scaling Reinforcement

rmax and rmin are computed according to:

$$rmax(t + 1) = \max\{rmax(t), r\} \quad (23)$$

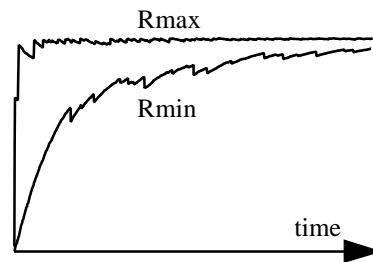
$$rmin(t + 1) = \min\{rmin(t), r\} \quad (24)$$

$$rmax(t + 1) = \lambda \cdot rmax(t + 1) + (1 - \lambda)r \quad (25)$$

$$rmin(t + 1) = \lambda \cdot rmin(t + 1) + (1 - \lambda)r \quad (26)$$

where  $\lambda$  is a positive number  $< 1$ . Equations (23) and (26) work together as follows. rmax increases as the system gets higher reinforcement values (equation (23)). rmin decreases as the system gets lower reinforcement values (equation (22)). Equations (25) and (26) allow the difference between rmax and rmin to converge, as the system learns, toward zero. rmin increases when low reinforcement values are infrequent (equation (26)). rmax is deliberately decreased in order to filter out large spurious reinforcement values (equation (25)).

Figure 11 shows graphically the expected behavior of rmax and rmin as the system learns.



*Figure 11: SSR convergence*

The algorithm's adaptation equations are

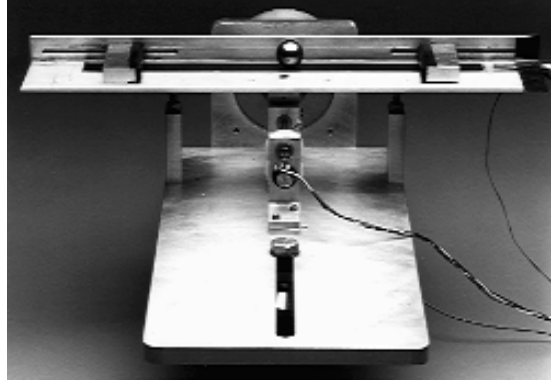
$$w_{\mu}(t+1) = w_{\mu}(t) + \alpha \hat{r}(y - \mu) \frac{\partial \mu}{\partial w_{\mu}} \quad (27)$$

$$\sigma(t+1) = \gamma \sigma + (1 - \gamma)(r_{\max} - r_{\min}) \quad (28)$$

where  $\gamma$  is a positive number  $< 1$ . It is used as an averaging factor.

When  $r = r_{\max}$ ,  $\hat{r} = 1$ . This is equivalent to having an LMS error  $e = (y - \mu)$ . As the system learns,  $(r_{\max} - r_{\min})$  converges toward zero, consequently,  $\sigma$  converges toward zero. The action becomes deterministic then.

SSR was applied with TD learning on a hardware ball balancer (Figure 12). The balancer learns to balance the ball in less than 15 min and keeps on balancing forever. [Benbrahim, 94].



*Figure 12: The Ball Balancer*

## 2.8 Eligibility traces

Eligibility traces [Barto, 83] are used to alleviate the problem of delayed rewards (there is no direct relationship between eligibility traces and Williams' characteristic eligibility). They are moving averages of the inputs and provide a decaying history that aids the algorithm in learning. The weights are updated using these traces instead of the inputs. Eligibility traces are computed as follows:

$$e(x, t + 1) = \lambda e(x, t) + (1 - \lambda)a(x)x \quad (29)$$

where  $x$  is the input,  $t$  is the time step,  $a(x)$  is the output and  $\lambda$  is the eligibility decay rate. All of the eligibility traces are reset to zero after every failure in order to reward only the actions that are responsible for that failure. The decay factor ensures that recent actions are rewarded more than older ones.

When using eligibility traces with SSR, equation (27) becomes

$$w_{\mu}(t + 1) = w_{\mu}(t) + \alpha \hat{r} e(t) \quad (30)$$

Where

$$e(t) = \lambda e(t - 1) + (1 - \lambda)(y - \mu) \frac{\partial \mu}{\partial w_{\mu}} \quad (31)$$

### 3 Neural Networks

Neural networks are models for function approximation. They are inspired by the brain's constitution of individual neurons. Each neuron has a specific function and many neurons interact with each other in order to execute complex tasks. There are many types of neural networks, some of which are more consistent with modern theories of neural circuit functions than others. They all are similar in the sense that they all use individual units configured according to the type of tasks the network is supposed to learn.

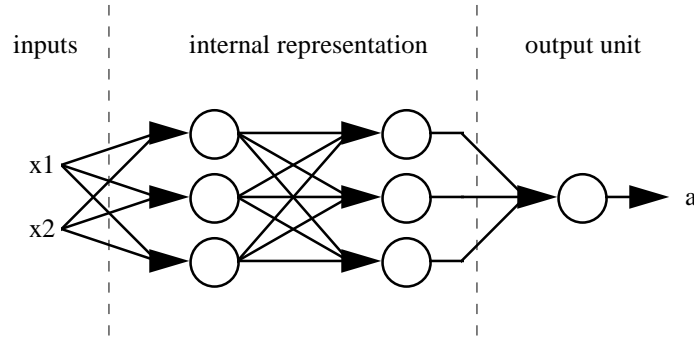


Figure 13: Artificial neural network

The artificial neural network in Figure 13 represents a generic network that contains an input layer, an internal representation (two hidden layers in this case) and an output layer. Each unit has a certain number of inputs and one output. The output is computed as follows:

$$y_i = f_i \left( \sum_{j=1}^n w_{ij} x_j \right) \quad (32)$$

where  $i$  is the index of the unit,  $x_j$  are the unit's inputs,  $y_i$  is the output,  $n$  is the number of inputs,  $w_{ij}$  are the unit's weights, and  $f_i()$  is a function that can be used to introduce nonlinearity to the system. The output unit is usually linear.

This is a very interesting architecture because it offers great flexibility in representing nonlinear functions. It is possible to represent any continuous function provided the network's organization and the nonlinear functions are chosen appropriately. The weights are updated in a direction that increases the system performance.

One can easily see from equation (32) that for large systems the number of parameters becomes exponentially large. This is a major drawback of neural networks, it makes them less practical for large scale systems in general, especially in real-time control applications.

### 3.1 Boxes

Probably the simplest form of neural networks uses the box configuration [Michie, 68] where the internal representation is replaced by "boxes". The input space is subdivided into equal size regions and each region is assigned to a box. The box's output is equal to 1 when the input lies within its designated region and is equal to zero otherwise.

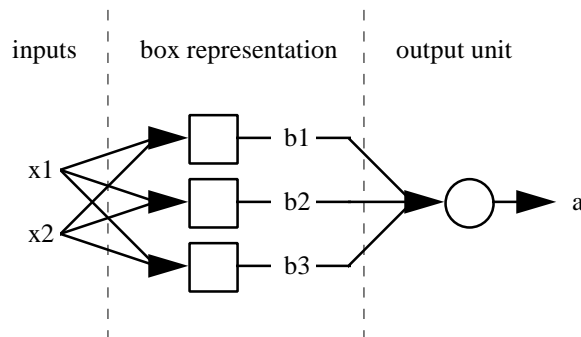


Figure 14: Two layer neural network using boxes

The output of the neural network in Figure 14 is computed as follows:

$$a = \sum_{i=1}^n w_i b_i \quad (33)$$

where only one  $b_i$  is equal to 1 and the rest are equal to zero. This network is very effective when high input resolution is not required [Benbrahim, 92], [Barto, 83]. The number of weights is equal to the product of the number of regions of each input. A major drawback of this network is that it cannot generalize its knowledge to regions of the input space for which it has not been trained.

The number of weights can be extremely large in the case of multidimensional inputs. In most applications many boxes never get visited. This results in a great waste of memory space. Hashing functions provide a reasonable solution to this problem. They map the large input space onto a smaller memory region. The most trivial hashing function maps the boxes on the first come, first mapped basis. The inputs that are left with no mapping memory space are either disregarded or assigned random box values.

### 3.2 Backpropagation

Backpropagation [Rumelhart, 86] uses the error signal of the output unit and backpropagates it to the units of the hidden layer. A typical backpropagation neural network is shown in Figure 15.

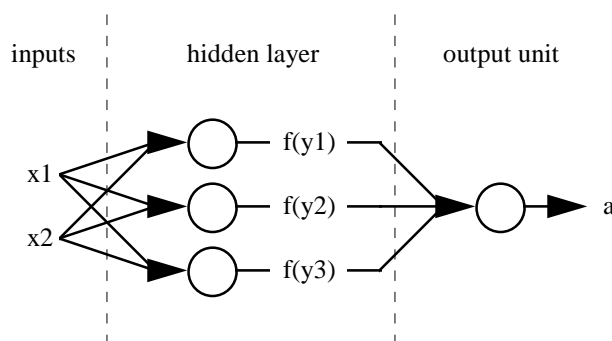


Figure 15: Two layer backpropagation artificial neural network

The output unit is linear and the hidden layer units are nonlinear using a nonlinear differentiable function  $f()$ . The sigmoid

$$f(y_i) = \frac{1}{1 + e^{-y_i}} \quad (34)$$

is the most commonly used nonlinear function in backpropagation because of its nice properties. It is defined everywhere, its output ranges from 0 to 1 and it is linear around zero and nonlinear on the edges. This last property allows for some units to learn linear behavior if needed.

The output of the network is

$$a = \sum_{i=1}^m v_i f(y_i) \quad (35)$$

where  $m$  is the number of hidden units,  $v_i$  are the weights of the output unit and  $f(y_i)$  are its inputs.

The weights  $v_i$  can be updated using Least Mean Squares (LMS) algorithm. The goal is to minimize the square of the error between desired and actual outputs  $\hat{a}$  and  $a$  respectively. LMS tends to minimize the following expression

$$E = \frac{1}{2} \delta^2 \quad (36)$$

$$\delta = \hat{a} - a$$

The output unit weights are updated as follows

$$v_i(t+1) = v_i(t) - \alpha \frac{\partial E}{\partial v_i} \quad (37)$$

$$v_i(t+1) = v_i(t) - \alpha \delta f(y_i)$$

where  $t$  represents the time step and  $\alpha$  is a learning rate ( $0 < \alpha < 1$ ).

The outputs  $y_i$  of the hidden layer units are computed as follows:

$$y_i = \sum_{j=1}^n w_{ij} x_j \quad (38)$$

where  $n$  is the number of inputs and  $w_{ij}$  are the weights of unit number  $i$ .

Using the LMS rule we obtain

$$w_{ij}(t+1) = w_{ij}(t) - \beta \frac{\partial E}{\partial w_{ij}} \quad (39)$$
$$w_{ij}(t+1) = w_{ij}(t) + \beta v_i f'(y_i) \delta x_j$$

Where  $\beta$  is a learning rate and  $f'()$  is the derivative of  $f()$ . The expression  $v_i f'(y_i) \delta$  is the back-propagated error.

The weights are updated according to the gradient of the LMS error; this is usually referred to as gradient descent.

### 3.3 CMAC

The Cerebellar Model Arithmetic Computer (CMAC) [Albus, 75] is an associative neural network that tries to mimic the biological sensory neurons in the brain's cerebellum. There are large numbers of sensors where each sensor is activated only when the input lies within a particular region. By using overlapping instead of distinct regions, the CMAC can generalize what it has learned about a certain region to its neighborhood.

A CMAC can be configured in several ways [Miller, 90]. The simplest form of CMAC is achieved by using several "boxes" configuration [Michie, 68] neural networks in parallel and averaging their outputs. The regions in each network must be shifted by a constant distance from each other. This provides the CMAC with overlapping regions and allows for generalization of the learning from one region to another. The individual networks can be called layers for simplicity.



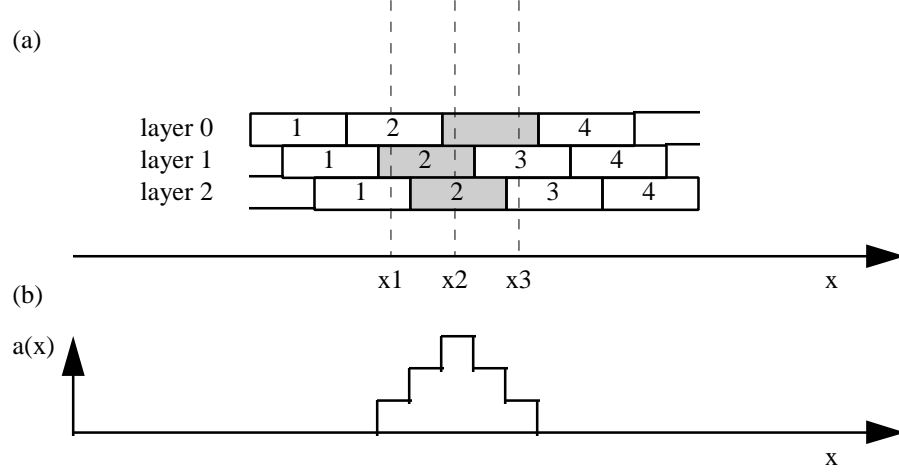


Figure 16: CMAC neural network input/output representation

The example of Figure 16(a) shows the state space partition in a three layer CMAC of a one dimensional input. The boxes of each layer contain the values of the learned actions  $a_{i,j}$  where  $j$  is the box number and  $i$  is the layer number. The output of the CMAC is equal to the average of the learned actions from each layer. The outputs corresponding to the inputs  $x_1$ ,  $x_2$ , and  $x_3$  are as follows:

$$\begin{aligned} a(x_1) &= \frac{a_{0,2} + a_{1,2} + a_{2,1}}{3} \\ a(x_2) &= \frac{a_{0,3} + a_{1,2} + a_{2,2}}{3} \\ a(x_3) &= \frac{a_{0,3} + a_{1,3} + a_{2,3}}{3} \end{aligned} \quad (40)$$

Suppose that we start with all of the weights set to zero. Then we present the CMAC with the input  $x_2$  exclusively and make it learn the optimum action  $a(x_2)$ . We then present the CMAC with the inputs  $x_1$  and  $x_3$  without performing any learning. The actions related to  $x_1$  and  $x_3$  will not be zero because of the boxes that they share with the input  $x_2$ . Figure 16(b) shows the output curve of the CMAC in the case where  $a_{0,3}=a_{1,2}=a_{2,2}=1$  and the rest of the weights are equal to zero. This example shows how the learned behavior from one region is generalized to neighboring regions even if they have never been visited.

The regions in the elementary CMAC configuration are of equal size  $S$  and each layer is shifted by  $S/N$ , where  $N$  is the number of layers. Using different size regions and irregular shifting distances can dramatically increase the CMAC's performance if done properly. Indeed, some regions in the input space being of more interest than others, may require

particularly high resolution for instance. The human eye, for example, uses high resolution at the center and very low resolution on the sides.

The CMAC's performance can also be increased by using the position of the input within the box in order to determine the importance of that box. In the example of Figure 16(a),  $x_2$  is located in the center of box number 2 of layer 2 while it is on the side of the other boxes. This might suggest that the output of layer 2 should be given more importance than the other layers. This can be achieved by using a weighted average of the outputs as follows:

$$a(x) = \frac{\sum_{i=0}^N w_i a_i}{\sum_{i=0}^N w_i} \quad (41)$$

$$w_i = \frac{S}{2} - d\left(\frac{S}{2}, \left(x - \frac{S}{N}\right)i\right) \text{ Modulus } S$$

where  $a_i$  is the output of layer number  $i$  and  $d()$  is the distance between the input and the center of the region where it lies in layer  $i$ . The weights  $w_i$  are at maximum if  $x$  is in the center of the region and zero if it is on the edge. The choice of the distance  $d()$  determines the shape of the output curve. A linear distance, for example can lead to a trapezoid shape and a quadratic distance will lead to a smoother shape.

This analysis can be easily applied to the case of multidimensional inputs. Special consideration, however, should be given to the generation of output weights. If a quadratic distance is used for example, the weights become too small near the edges and slow learning occurs. One way to alleviate this problem is to compute the weights for each dimension as described above and choose the smallest one as the final weight, [Miller, 90].

## 4 Outstanding challenges

Even though learning methods have been more successful than conventional control methods in solving many problems, there are still some serious outstanding challenges. This does not mean that conventional control methods are adequate enough to deal with these challenges either. Two of the challenges that if overcome can greatly improve learning capability are sensor input range and resolution adaptation, and learning from analogies. Most learning methods can partially deal with these issues, but there has not been any significant breakthrough yet.

## **4.1 Sensor adaptation**

The issue of input selection is very crucial. If a learning system, such as a neural network for example, can learn to identify the relevant input regions, then it can concentrate all of its resources on those regions. Furthermore there are many input regions that require higher resolution than others. In balancing a broom stick, for example, it is not necessary to accurately measure the angles that are far away from the vertical. The angles near the vertical, however, are very important for stability. When using multi-dimensional input vectors, there can be many combinations of inputs that never happen. Using the same balancing example, it is hard to conceive of the stick moving at high positive angular velocity if it is at a large negative angle from the vertical.

Humans have a very good sensor adaptation system. A person can by experience develop a sense for things in which there is a special interest. There are many examples of such things: music, food and wine tasting, perfume, pool, etc. The brain also develops special cells that deal with these senses as needed: a child exposed to music will have extra sound sensing capability.

Genetic algorithms and hashing functions have shown promising results in dealing with these issues. Hashing functions can be used to select relevant inputs, and genetic algorithms can generate sensing capability as needed.

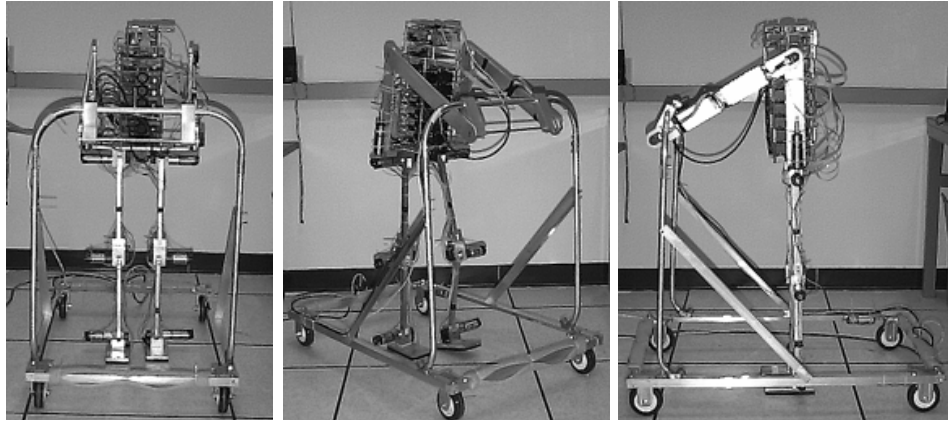
## **4.2 Learning from analogies**

Learning from analogies is one of the most important features that allow animals to learn complex tasks. In order to solve a problem, a wise person breaks the problem up into small parts and solves each one individually by using analogies. A trivial example is using a stairway to go up to a higher floor. Another example is proving a theorem.

If a learning system can grasp the notion of analogies then it can learn to identify subtasks that are analogous to what it has learned previously. It can then build a modular architecture that can solve many problems and that can learn new tasks quickly.

## CHAPTER FOUR

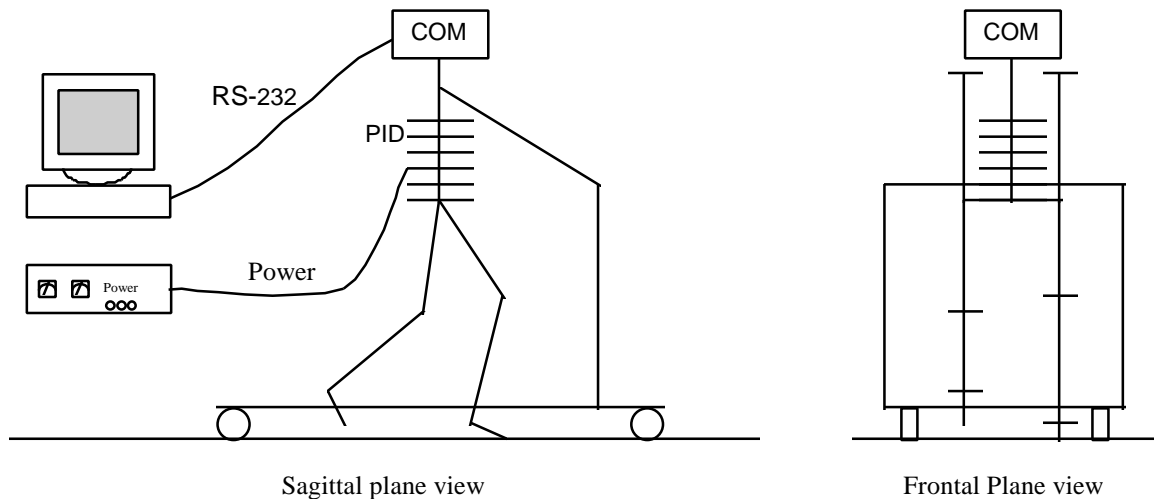
### THE BIPED



*Figure 17: Biped views*

#### 1 Mechanical structure

The biped (Figure 17, Figure 18) is an aluminum six joint, seven link biped robot. It has two accessory arms with no elbows; the shoulder and hand joints are free, and not powered. The robot is restrained to the sagittal plane by pushing a baby walker-like cart. The only contact between the robot and the cart is at the hands. The six leg joint actuators are powered with DC motors, and joint angles are measured using optical encoders. All joint axes of rotation are orthogonal to the sagittal plane.



*Figure 18: The Biped*

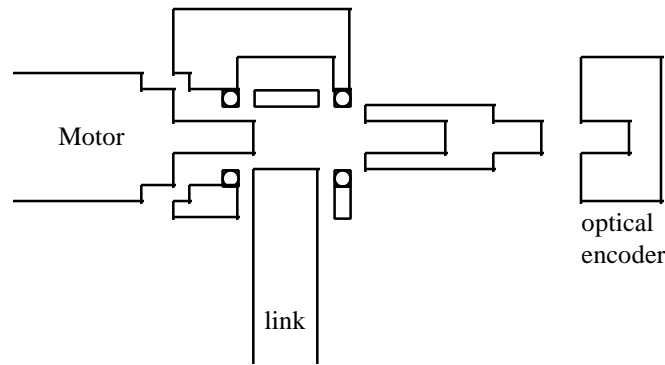
Each leg joint is powered by a DC motor with low gear ratios, 1:17 for the ankle, 1:27 for the knee, and 1:50 for the hip. Using low gear ratios has several advantages that dramatically increase the robot's performance and make it closer to animal walking than when using high gear ratios.

DC motors cannot control speed and force independently as well as biological muscles do. High gear ratios have a constant stiffness that seriously limits force controllability. Many of the natural movements that humans perform during walking can be achieved by using low gear ratios. Indeed, low gear ratios allow free swinging of the legs and taking advantage of gravity, thus giving smooth and natural looking movements. They also allow for the joints to act as springs. The springiness can be easily controlled using PID controllers. The problem of backlash inherent to all gear heads is reduced when using low gear ratios. The advantage to using gears at all is that they increase the torque at the joint.

A major disadvantage of using low gear ratios is that the torque is very low. The torque grows proportionally with the gear ratio. In order to obtain the necessary torque larger thus heavier motors need to be used. A compromise must then be found between the motor's weight and torque. This seriously hinders the biped's controllability, and forces the walking to be very conservative in energy. Iron-less DC motors (Escap 35NT2R-R32 by Portescap), which have a high torque to weight ratio, were used, in order to make up for this loss of torque.

Joint positions are measured with optical encoders, with a resolution of 2000 counts per revolution (HEDS-5540 by Hewlett Packard). Each optical encoder is directly attached to the joint axle instead of the motor's axle. This eliminates the error that is generally introduced by gear heads.

Each joint contains ball bearings to minimize friction and sideways movement. This is done because it is critical to have accurate and reliable measurement of joint positions. Any sideways movement can seriously hinder the biped's stability, especially since it is, by design, incapable of doing any movement to correct sideways imbalance. Figure 19 shows a disassembled view of a joint.



*Figure 19: Joint structure*

The feet are padded with rubber pads (computer mouse pads) in order to maximize foot adhesion to the floor, and dampen foot impact on the floor. Pressure sensors are placed between the foot and the pad, on the front and on the back of each foot. These sensors can be used to determine the front/back balance. The most appropriate sensors, we found, for this application are flat flexible sensors each of which is composed of a resistive and a conductive surface put together in parallel (FSR by Interlink Electronics). Once the conductive surface is pressed against the resistive surface, the resistance measured between the extremities of the two surfaces is proportional to the applied pressure.

The body of the biped consists of a stack of PID controllers and a local processor board. Extra weight is added to the body in order to stabilize the walking. Indeed, because of the moment conservation law, leg movements induce body movement. These movements are inversely proportional to the respective weights. A biped with a heavy body and light legs is thus more likely to be stable than a biped with a light body.

The walking cart is made of a light aluminum rectangular structure, a handle, and four wheels. The main purpose of the cart is to keep the robot from falling sideways. Adding extra weight on the corners of the cart's base increases the cart's moment of inertia, thus prevents it from rotating.

The robot's hands can rotate freely around the cart's handle, and the shoulders can also rotate freely. Because these joints are free and passive, the cart does not help the robot keep its front/back balance. Since the cart's handle is always at a constant height, and the

wheels are on the floor, the robot's height and posture can be determined using optical encoders that measure the hand and shoulder joint angles.

## 2 Mechanical specifications

The robot is 1 m high, weighs about 9 kg, and the foot area is 15.2x7.6 cm. The arms weigh 0.4 kg each and the cart weighs 4 kg, where most of the weight is concentrated on the corners of the base. The cart's handle is 0.76 m high and the dimensions of the base are 1.2x0.6 m.

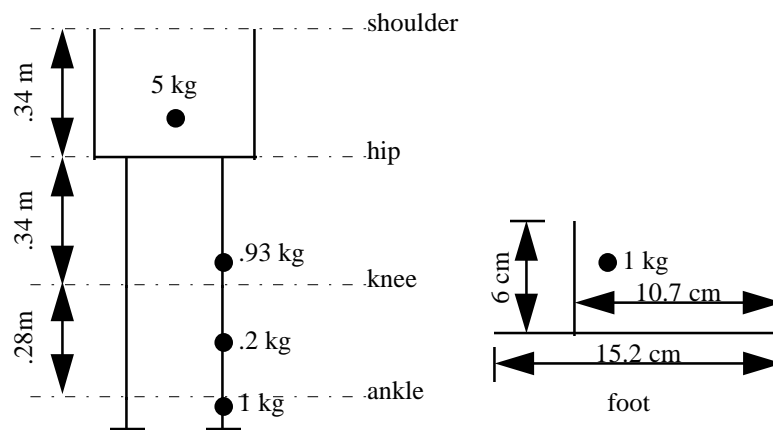


Figure 20: Mechanical specifications

Figure 20 shows some of the biped's mechanical specifications. The centers of gravity and weights are represented by the dark dots. The motors that actuate the hip, knee and ankle joints are mounted on the body, the thighs, and the feet, respectively.

## 3 Kinematics

A major advantage of using the cart is that it allows us to determine the position of every joint and every point in the robot relative to the cart's position. These measurements are only significant as long as the cart keeps its four wheels on the ground.

Some of the variables that are most important in controlling the robot's walking are: body posture, body height (hip height), and foot positions. Using the notation of Figure 21 we compute the posture  $p$ , the body height  $y$  and the foot position  $x_f$  and  $y_f$  as follows:

$$p = w + s - \mathbf{p} \quad (42)$$

$$y = L + b \sin(w) + c \sin(w + s)$$

$$x_f = b \cos(w) + c \cos(w + s) + d \cos(w + s + h) + e \cos(w + s + h + k)$$

$$y_f = b \sin(w) + c \sin(w + s) + d \sin(w + s + h) + e \sin(w + s + h + k) - FH$$

Where  $FH$  represents the foot height. Since the feet are supposed to always be flat, the angles associated with the term  $FH$  are equal to  $\pi/2$ . Notice that all of the variables are absolute except for the foot  $x$  position. There are no sensors that measure the cart's horizontal motion. This is not a problem because as far as the walking is concerned, only the position relative to the body is of interest. However, it is easy to compute an absolute  $x$  position since there is always at least one foot on the ground, and there is no foot slippage. The cart's horizontal movement can be determined relative to the supporting foot position. The  $x$  offset can then be incremented by the step length every time a new step is taken.

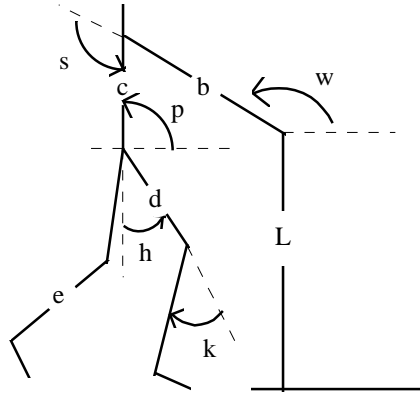


Figure 21: Biped kinematics

## 4 Control structure

The biped uses six PID controller boards to control joint positions; and a local processor to communicate with a remote PC (486-66Mhz), and perform low level processing, (Figure 22). The PC communicates with the local processor using RS232 at 38.8 kbps. It reads sensor readings and sends the new desired joint positions. These desired positions are generated by the learning algorithm that runs on the PC.



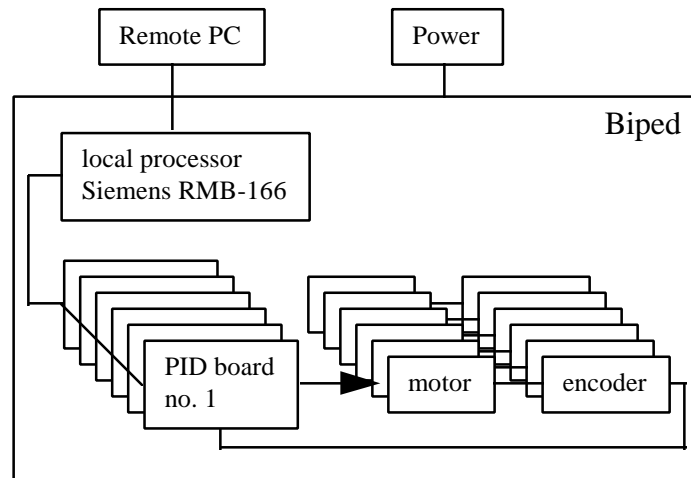


Figure 22: Control structure

The PID control boards Figure 23 use the LM628 PID controller chip. Each board contains an address decoder, an LM628 PID chip, an 8 bit digital to analog converter, a high power operational amplifier (LM12), an 8 Mhz oscillator, and +/- 5V voltage regulators. All of the boards have their data and address buses connected to the local processor board in parallel.

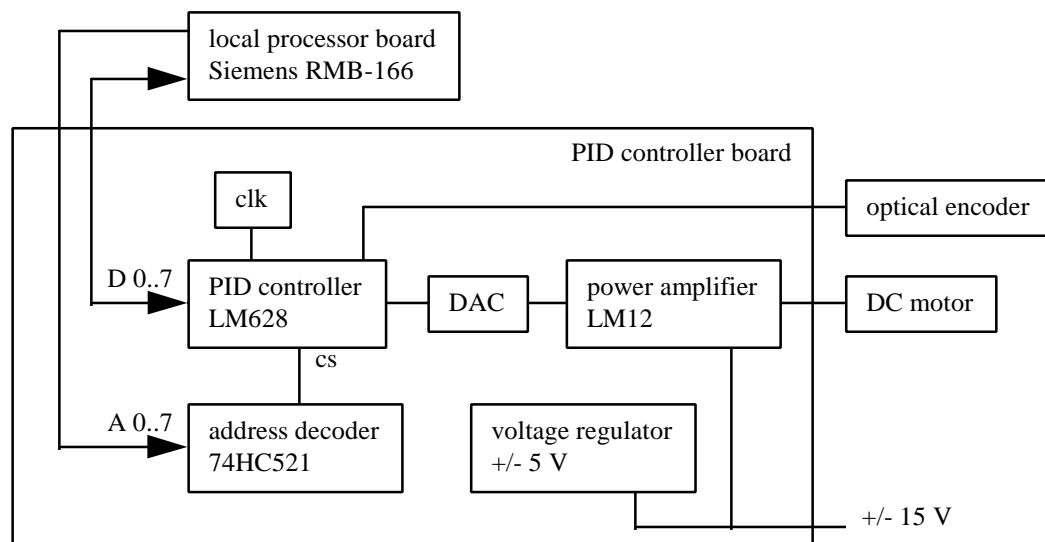


Figure 23: PID controller board

There is also a buffer and a transceiver that are used by data and address busses between the PID and the processor boards.

The PID boards are completely independent from each other. This alleviates the problem of electrical noise, and makes it easy to upgrade the biped and increase the number of joints.

The LM628 chip offers great flexibility in performing PID control by using many programmable parameters. There are filter parameters and trajectory parameters. Filter parameters are the PID gains, the maximum allowable value of the error integral sum, and the sampling rate used to compute the derivative error. Trajectory parameters are the acceleration and speed used to reach the target position. These parameters mainly affect the shape of a joint trajectory. Using a low acceleration, for example, generates smooth trajectories. Once all of the parameters are set the chip continuously performs PID control. Any of the control parameters as well as the desired position can be changed in real-time. The control sampling period is 256  $\mu$ s.

This chip offers many data reporting commands that allow the user to monitor closely the control process. The data that are of most interest to the learning algorithm are joint position and joint velocity. Thanks to the optical encoders and high speed clock, the data are very accurate and reliable.

## **5 Communications interface**

The local processor board is the intermediary between the PC and the rest of the biped. To minimize the communication time with the PC, it waits until it receives the new desired joint positions. It then sends sensor data it had previously collected, mainly joint positions, joint velocities, and foot pressures. When not communicating with the PC, it transmits the new desired trajectory data to the PID boards using the low level protocol required by the LM628 chip, and reads sensor data.

This interface provides great time saving for the main PC, especially in serial communication time. It can also handle some simple walking control tasks, like keeping the feet always flat and determining the biped's posture and body height. It also makes the robot control more robust because the control timing is more tightly managed this way.

## CHAPTER FIVE

### THE LEARNING ARCHITECTURE

This chapter describes the learning architecture that was developed as part of this research. Section 1 describes the general approach of this architecture as well as the different issues that it tries to deal with. Sections 2 and 3 describe the different modules that constitute the architecture. Section 4 describes the reinforcement learning that takes place within the architecture and shows some of the learning equations. A thorough description of the learning algorithms used in this architecture is presented in Chapter 3.

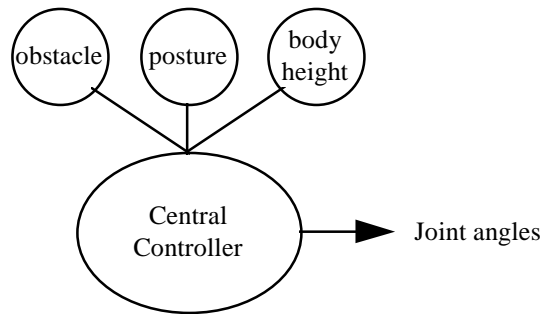
#### 1 General approach

The ultimate tangible goal of this research is to achieve dynamic walking with a high level of adaptation and with minimum knowledge about the robot's dynamics. There are two major motivations behind this choice. First, it is too complicated to find precise knowledge about the robot, and we do not know how to use this knowledge accurately in order to achieve desired walking patterns. Second, we know that humans acquire great walking ability partly because they keep constantly on learning how to optimize their walking behavior, and they always adapt to new terrain. It is then a great incentive to try to go in this direction for the biped robot. Learning and adaptation methods have been tried on real biped robots and showed promising results [Miller, 94],[Zheng, 90].

A powerful learning architecture should be able to take advantage of any available knowledge [Franklin, 89]. Indeed, there are some situations where a simple rule or a simple linear controller can achieve the desired task. One method is to use different controllers in parallel, and a switching mechanism that activates the appropriate controller [Jacobs, 91], [Narendra, 94], [Sklansky, 66], [Widrow, 60]. Franklin [Franklin, 88] uses a hybrid method where a learning controller refines the control of a fixed controller.

Controller switching is a very powerful method, yet, it has a major disadvantage. Once a controller is activated, the system does not always benefit from the other controllers' experience. Some switching mechanisms avoid completely excluding the other controllers by using a weighted average of all outputs. Determining the appropriate weights, however, requires a complex switching mechanism, especially if those weights are functions of the inputs.

In this study, instead of a complex switching mechanism, a “melting pot” is used. The melting pot is a central controller that uses the experience of other controllers in order to learn an average control policy. This centralizes the common knowledge of all controllers in one, thus alleviating their burden. The central controller controls the robot in nominal situations, and the peripheral controllers intervene only when they consider that the central controller’s action contradicts their individual control policies (Figure 24). The action is generated by computing the average of the outputs of all controllers that intervene including the central controller. Each peripheral controller’s role is to correct the central controller’s mistakes and issue an evaluation of the general behavior. The central controller then uses the average of all evaluations to learn a control policy that accommodates the requirements of as many peripheral controllers as possible.



*Figure 24: Architecture*

The learning architecture used here is based on the assumption that there exists a nominal behavior governed by a minimal number of inputs. The deviations from this behavior are supposed to be small and caused by perturbations. Peripheral controllers individually act according to these perturbations. This is similar to decoupling a multi-variable function, except that the decoupling occurs around an operation point instead of over the whole input space.

This architecture is also inspired by the Taylor expansion of a multi-variable function around an operation point. Equation ( 43) shows a first order approximation of a Taylor expansion of a function  $f(x,y)$ . If the deviations  $dx$  and  $dy$  are small, then this equation can be sufficiently used to represent the function  $f(x,y)$ . The first term can be loosely considered as the central controller and the derivative terms as the peripheral controllers. A rigorous use of this equation by the learning architecture would definitely achieve a powerful controller. However, this would require complex central and peripheral controllers. This equation is then used just as an inspiration model.

$$f(x_0 + dx, y_0 + dy) \approx f(x_0, y_0) + \left( \frac{\partial f(x, y)}{\partial x} \right)_{(x_0, y_0)} dx + \left( \frac{\partial f(x, y)}{\partial y} \right)_{(x_0, y_0)} dy \quad (43)$$

The controllers proposed here can be achieved by using different kinds of control systems. The central controller as well as some of the peripheral controllers in this study use adaptive CMAC neural networks. Other peripheral controllers for which the desired control policy is known in advance are fixed.

Most complex controllers here use CMAC neural networks,. The peripheral controllers can use learning and adaptation methods, or can be fixed.

### 1.1 Modular architecture

It is in theory possible to conceive a single neural network that can use as inputs all the variables that determine the state of the biped's walking and learn an optimal control policy. When the number of inputs is very large, as is the case with biped robots, the neural network becomes exponentially large. Large neural networks learn very slowly, they use large amounts of memory, and they cannot be effectively used in real-time applications because they are very slow.

Guided by the same idea used in decoupling complex dynamic equations, it is beneficial to use several neural networks with smaller numbers of inputs, instead of one large neural network, whenever it is possible. A “boxes” configuration neural network that has three inputs subdivided in ten regions each, for example, contains  $10^3$  weights. If it is possible to use three neural networks with one input each instead, then the total number of weights would be only 30.

Unfortunately it is not always possible to decouple the robot's control as described above. Indeed the biped's state variables are tightly dependent on each other. However, using the assumption that there is a nominal behavior and that the deviations from it can be small, then it can be reasonable to assume that when dealing exclusively with these deviations, the inputs are independent and can be controlled separately. If the robot's posture deviates from the normal value by a small amount, correcting it would not affect the body height for example.

### 1.2 Knowledge incorporation

Humans use different types of knowledge in order to walk and keep their balance. They use visual information, acceleration measurements, foot pressures, special knowledge about the terrain condition, etc. Not all of this information is necessary, but each type of

information adds a great contribution to the walking process: try to walk with your eyes closed.

The learning architecture used here tries to incorporate as much information as possible in one homogenous structure. The information need not be all of the same kind. Each piece of information can be used as a peripheral controller. The central controller constantly learns from that information, and uses it as necessary.

There are two ways information can be incorporated. First, it can be used to affect the control action by adding its contribution to the total output. The central controller, then, learns from that example. Second, a peripheral controller can issue an evaluation of the action. If the action causes a deviation in the posture, for example, the posture controller will then issue a bad evaluation. The central controller updates its weights in a direction that improves the evaluations. Peripheral controllers can intervene either at the action or evaluation levels, or both.

### **1.3 Task definition**

Task definition is a very important question in learning and control. Before controlling a system it is necessary to know what is expected from it. A DC motor's task, for instance, is to turn at a certain speed or go to a certain position. This task can be efficiently performed using a PID controller. Biped walking, however, is much harder to define.

Most of the actual research tries to solve the problem of biped locomotion by answering two main questions: which variables best reflect the biped's behavior, and how can these variables be controlled in order to achieve desired walking patterns. This approach can be effective if the walking pattern is well defined according to these variables, and if it is possible to control them efficiently. Because of the complexity of biped dynamics, only a few of these variables are used, therefore, the walking behavior is only partially defined. This forces the biped controller to choose solutions that are not necessarily optimal.

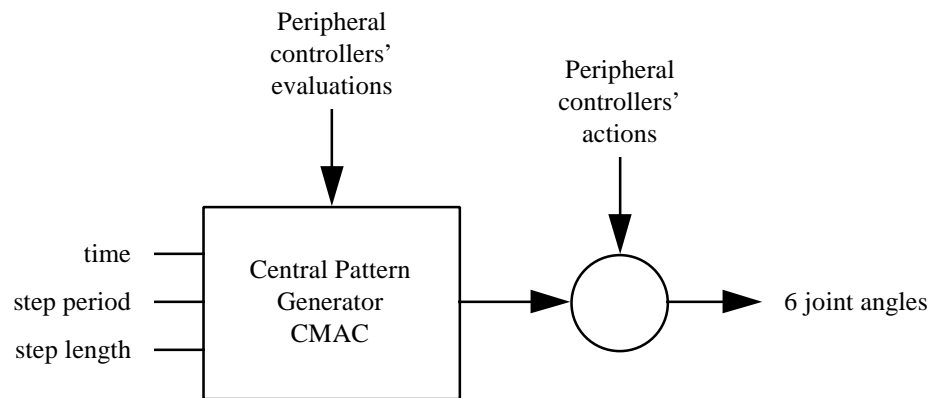
Partial constraint satisfaction is a very powerful method that is widely used in artificial intelligence [Freuder, 92], [Jordan, 92]. The desired walking pattern can be defined by a set of constraints. The robot learns to execute movements that respect all of these constraints. If, for instance, we want the robot to walk in an upright position, not to drag its feet and to have elegant walking, then it must respect the following constraints: keep the body height and posture within a predefined range, keep the free leg above a certain height, have periodic movements, and put a cap on energy spending. By respecting these constraints, the walking robot tend to walk as desired.

These constraints do not completely specify the walking pattern. They specify only the boundaries that cannot be crossed. The biped controller is free to choose any solution within these boundaries. It is then given a chance of finding an optimal solution. The free area can be further reduced as more task defining constraints are used.

Special care should be given to the choice of constraints. They should not be too restrictive in order to avoid conflicting interests. The robot constantly tries to find a compromise between all of the constraints. Some constraints can be given priority over other constraints as long as this can improve the walking or resolve conflicts.

## 2 Central Controller

The central controller (Figure 25) is represented by a Central Pattern Generator (CPG). The CPG continuously generates joint positions according to the desired walking pattern. The CPG uses a CMAC neural network to learn optimal joint positions. The inputs to the CMAC are time  $t$ , step length  $S$ , and walking period  $T$ . Assuming that an appropriate CPG has been learned and that there are no perturbations, this should be sufficient to achieve the desired walking. It is equivalent to an open-loop controller or voluntary motion [Zheng, 90]. The small number of inputs allows the use of a small CMAC thus speeding up the learning and execution time for action generation.



*Figure 25: Central controller*

It is, of course, inconceivable to imagine a world with no perturbations. However, even though the CPG is not designed to react to individual perturbations, because of its small number of inputs, it is forced to learn the most robust solution. An example of such a solution is locking the knee joint of the supporting leg. This in general provides stable walking and keeps the robot within a certain height.

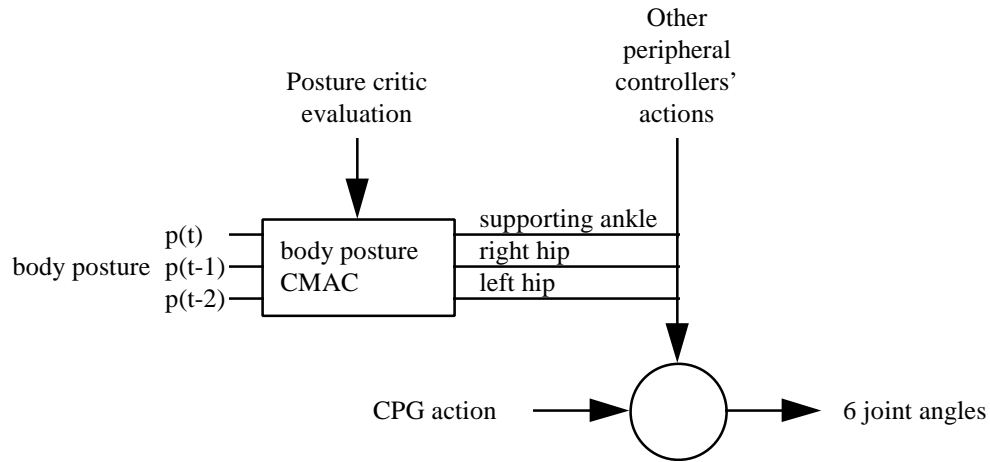
The CPG uses reinforcement learning in order to learn an optimal policy. The CMAC weights are updated using the reinforcement signals received from the constraint guardians, also referred to as peripheral controllers. The CMAC also learns from the corrective actions that these controllers generate when their respective constraints are violated. Learning from these actions can be achieved using supervised learning where the peripheral controllers act as teachers, or incorporate the action in the reinforcement

learning algorithm. The choice between the two approaches is determined by the reliability and accuracy of each peripheral controller. The learning equations are presented in Section 4.

### 3 Peripheral controllers

#### 3.1 Body posture

Posture is the most critical controller in biped walking. Indeed, small deviations from the vertical can lead to very large torque on the supporting foot. Usually when the posture is beyond a certain limit or changing at large speed the robot becomes hopelessly unstable. Furthermore, correcting the posture abruptly can also be fatal. The posture controller (Figure 26) acts on both hip and ankle joints.



*Figure 26: Body posture peripheral controller*

Ankle joint torque can sufficiently control the posture only when the projection on the ground of the center of gravity of the biped is on, or very close to, the center of the support area, and when the posture is changing slowly. Notice that in theory if the ZMP lies outside the support area, then the ankle torque has no effect on stability. In order to better see how well the ankles can control posture, put your feet together and see how far you can lean forward before falling over. Usually one cannot lean too far, relying only on the ankles to ensure stability.

In order to recover from large posture deviations, humans usually use the free foot to stop the fall. By choosing the appropriate foot position, the robot can efficiently recover from large deviations. Controlling both hip joints can be a reasonable substitution to this behavior. Indeed, there is a direct relationship between hip joints and foot positions. Foot



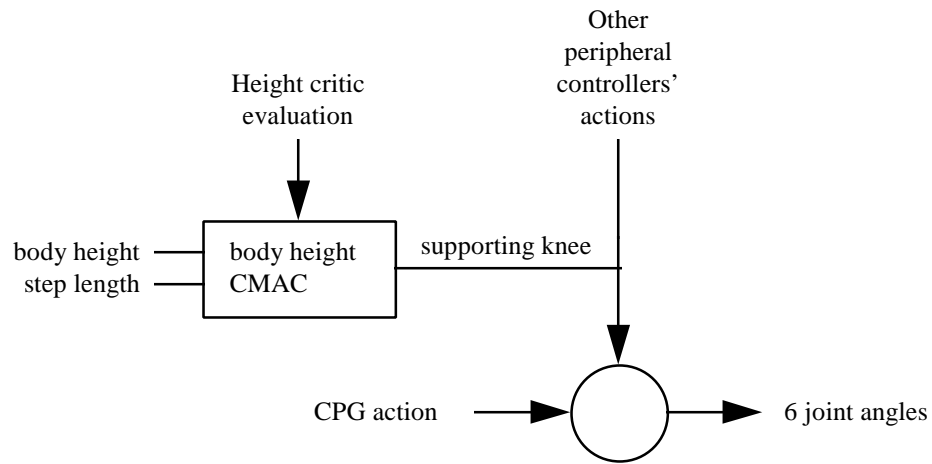
height is not of major concern here because it is individually controlled by other controllers.

Choosing joint instead of foot position control relieves the extra computational burden of using inverse kinematics to determine the appropriate joint positions. Furthermore since the CPG generates joint positions, adding posture control can be straightforward.

The posture controller uses a CMAC neural network. It uses the posture values at the current and two previous control cycles as input to generate three outputs: supporting ankle joint, and two hip joints. Using the same input, a similar CMAC forms part of the CPG critic (Figure 31). Both CMACs use reinforcement learning with TD learning as described in Section 4.

### 3.2 Body height

The body height controller (Figure 27) determines the knee joint angle of the supporting leg. Because of the biped's heavy weight and high damping factors, the height changes are relatively small. A simple proportional controller can be sufficient, in general.



*Figure 27: Body height peripheral controller*

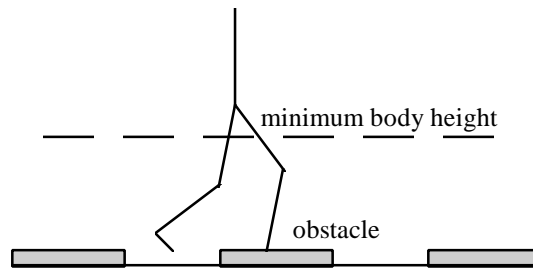
The CMAC's inputs are step length and body height. The weights are updated using reinforcement learning. Since the right body height is unknown in advance, the learning algorithm uses the biped's global reinforcement, the one used by the CPG. The optimal body height is a function of the global walking performance. It is also indirectly related to the power of the DC motor. There are many situations where the motor is not strong enough to change the body height.

There is an overlap between the body height controller and CPG functions, since the CPG is supposed to learn an optimal body height on its own. The body height controller is used here mainly to speed the learning up and set some absolute height limits. These limits can be predetermined based on reasonable walking patterns.

The body height controller also uses a CMAC to generate a critic for the CPG and body height action CMACs. The inputs are the same as for the action CMAC. The weights of the critic are updated using TD Learning as described in Section 4.

### 3.3 Step constraint

To make the robot move forward and prevent it from dragging its feet, imaginary obstacles are put on the ground (Figure 28). The distance between these obstacles is equal to the desired step length minus the obstacle length. The obstacles also move at a speed equal to the desired walking speed, functioning as a tread-mill. The height of the obstacles is a predefined constant with a reasonable value. The length is a simple linear function of the desired step length.



*Figure 28: Step constraint*

There is no peripheral controller that handles this task in particular because it is relatively complex. It is mostly the CPG's responsibility. However, when the robot hits an obstacle, the step height guardian generates a failure signal that the CPG incorporates in its learning. There is no particular critic either because it is hard to find a variable that quantifies how well this constraint is respected without being too restrictive.

### 3.4 Total energy

Energy spending is not a necessary control variable in achieving dynamic walking. However, a biped that uses minimum energy is bound to keep constant height and posture, and uses smooth movements. Such a walker is more likely to be dynamically stable than a walker that spends unnecessary energy.

As is the case with the step height constraint there is no simple action that can be taken to minimize energy spending. This constraint guardian generates a reinforcement of the CPG's action based on the amount of energy spent.

## **4 Reinforcement learning**

Since the walking task is defined only by a set of constraints, there can be many possible walking patterns. Most of these solutions, however, are not optimal. It is thus necessary to be able to explore the action domain and choose an optimal solution that is a compromise between all of the constraint requirements. This approach provides great flexibility in shaping the walking pattern.

Reinforcement learning is well suited for this kind of application. The system can try random actions and choose those that yield good reinforcement. While searching the action space, the system updates its weights in order to find a compromise between all constraints.

The learning algorithm searches the action space using a Stochastic Real Valued (SRV) unit at the output [Gullapalli, 90]. The unit's action uses a Gaussian random number generator. The mean of the Gaussian is determined by the controllers' outputs, and the standard deviation is determined by the Self Scaling Reinforcement algorithm (SSR), [Benbrahim, 94]. The reinforcement signal is generated using TD Learning [Sutton, 84, 88] and the SSR algorithm. These learning algorithms are thoroughly described in Chapter 3.

### **4.1 The actor**

The actor's action is a vector of six joint positions that are directly sent to the robot. The action is generated by a Gaussian random unit. The mean of the unit is determined using the average of the outputs from all controllers. Peripheral controller contributions are zero, and are not taken into consideration while computing the average, unless their respective constraints are violated. The performance can in general be improved using a weighted average, provided the relative importance of each controller is known.

When none of the constraints are violated only the CPG's output is taken into consideration. Once the system has learned an optimal policy, the standard deviation of the Gaussian converges toward zero, thus eliminating the randomness of the output.

Figure 29 shows the actor's configuration. Note that the inputs and outputs of the linear and Gaussian units are six dimensional joint vectors. The outputs of the controllers are all joint angles.

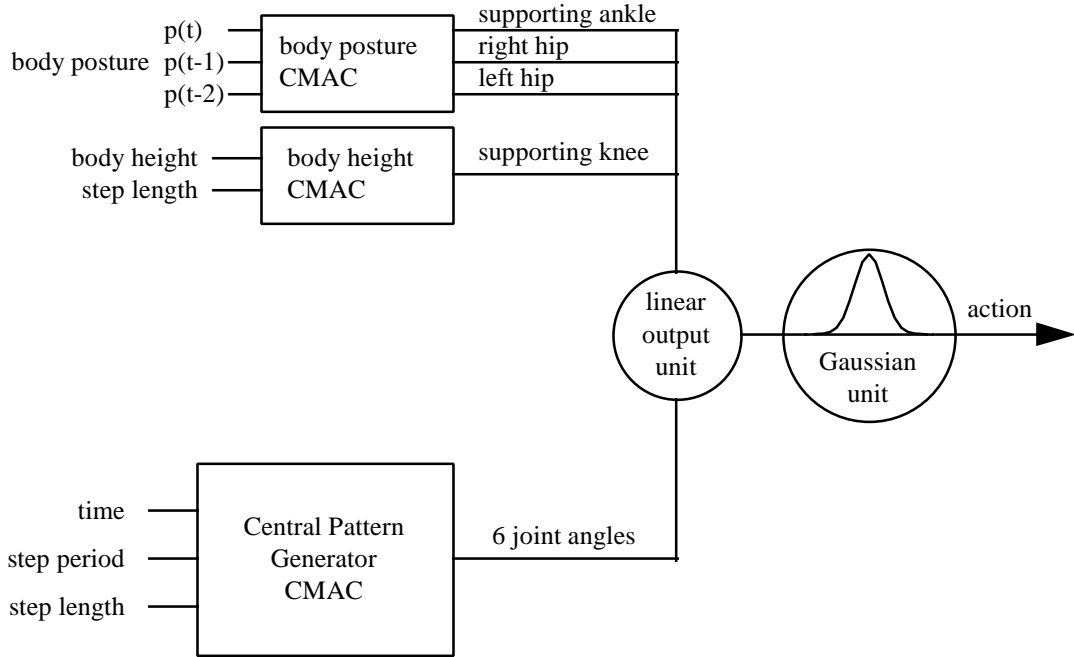


Figure 29: The actor

The CMAC weights of all of the controllers including the CPG are updated using the same equations (equations: (44), (45)). Figure 30 shows the actor configuration of Figure 29 with focus on only one CMAC.

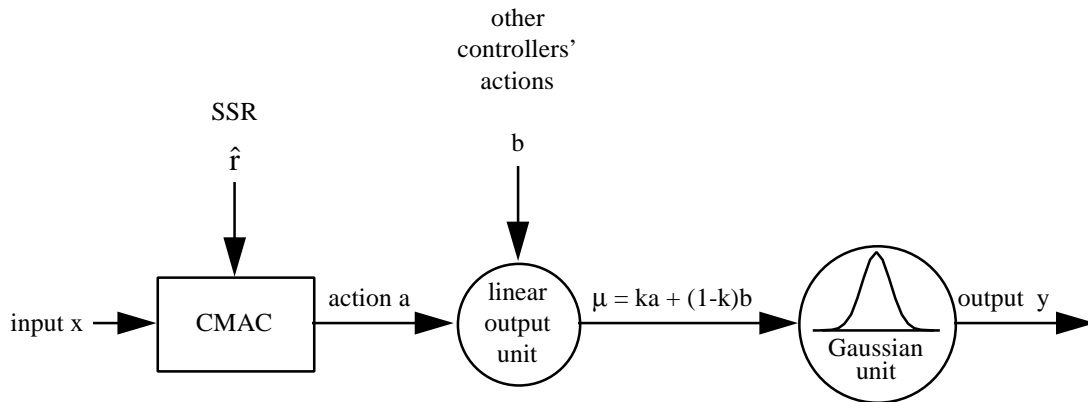


Figure 30: General actor configuration

The weights  $w$  of the CMAC and the standard deviation are updated as follows:

$$w(t+1) = w(t) + \alpha \hat{x}(y - a) \frac{\partial a}{\partial w} \quad (44)$$

$$\sigma(t+1) = \gamma \sigma + (1 - \gamma)(r_{\max} - r_{\min}) \quad (45)$$

These are the same equations as the SSR equations of Chapter 3.

The weights are updated in a direction that moves the action  $a$  towards the output  $y$  if the reinforcement is positive, and away from  $y$  if the reinforcement is negative. Since the output  $y$  combines all the controllers' outputs, the CMAC learns by taking example from their actions as well as from the reinforcement signal. This combines at the same time supervised learning with reinforcement learning. A teaching peripheral controller can thus be easily included here.

This behavior can be better understood by considering the following case. Let us suppose that there is only one extra controller and that the standard deviation as well as the factor  $k$  of Figure 30 are zero. Equation (44) becomes then

$$w(t+1) = w(t) + \alpha \hat{x}(b - a) \frac{\partial a}{\partial w} \quad (46)$$

This equation shows that the CMAC learns directly from the other controller. If this latter is a perfect controller, then the reinforcement is always equal to 1. This then becomes a simple case of supervised learning using LMS.

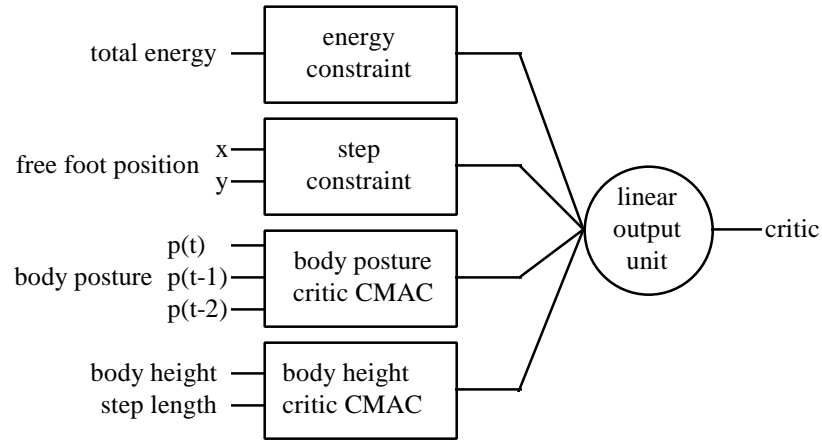
It is possible to make the CMACs learn independently from each other by using separate random output units and separate reinforcement signals. This means that there should be a Gaussian unit after each controller and no Gaussian unit after the linear output unit. Each controller would be updated using the difference between its action and the output of its Gaussian unit.

## 4.2 The critic

The critic is the most important factor used to reconcile different constraints. Each constraint guardian issues a critique of the robot's action when that action has a direct effect on the constraint's control variable. The energy constraint, for instance, monitors only the total energy.

The critic used to train the CPG is equal to the average of the constraint guardian critics. The maximum value of the CPG critic can only be reached if the system performs well according to all or most constraint guardians. The CPG critic is thus a global performance measure. As in the case of the action, a weighted average could be used to enhance the performance.

While in the learning process, the CPG tries random actions and updates its weights in a direction that increases the CPG critic's evaluation. The CPG's learning, thus, takes into consideration all of the constraint control variables.



*Figure 31: The CPG critic*

Figure 31 shows the CPG critic configuration. The posture and height critics use CMAC neural networks. These networks are trained using TD learning as described in chapter 3. Each critic generates a reinforcement signal as follows:

$$\hat{r} = r + \gamma p(x_k, t) - p(x_{k-1}, t-1) \quad (47)$$

where  $r$  is the raw reinforcement and  $p(x, t)$  are the outputs of the CMACs (Figure 8).

Energy and foot step critics are predefined. The energy critic generates a reinforcement value that is simply proportional to the negative of the energy and the foot step critic generates a simple failure/success reinforcement. This reinforcement is equal to -1 if the free foot hits the obstacle and zero otherwise.

The total critic is generated through a linear output unit that computes a weighted average of the individual critics. Total energy and step constraint critics are assigned very low

weights because they do not provide a prediction of success and do have state evaluation information. Their contribution to the total critic is chosen to be less than 5%.

## CHAPTER SIX

### EXPERIMENTS & RESULTS

#### 1 Implementation issues

Even though reinforcement learning methods have proven to be able to control complex systems using minimum knowledge about their dynamics, they are impractical in real-time applications because of the long time it takes them to learn. Also many systems cannot sustain the stress caused by multiple failures (i.e. a beginner trying to learn ice skating). It is thus necessary to speed the learning up by pre-training the neural networks and providing safeguards.

The CMAC neural networks used in the biped's learning are pre-trained using a biped walking robot simulator [Latham, 92] and predefined simple walking gates. The simulator's parameters have been modified in order to fit the hardware robot as closely as possible.

The learning architecture used here allows for the use of parallel controllers. These controllers function both as safeguards and as teachers. The biped also starts in a standing position and takes small steps ensuring that the assumption of nominal behavior and small perturbations, on which the learning architecture is based, remains valid.

The biped is a highly unstable system. Using high magnitude random actions can throw it out of balance. Action generation is thus a very important issue. It is dealt with by using a low pass filter at the robot joint level, and generating relative actions instead of absolute actions, assuming that the nominal action is a continuous function. This issue is described in more detail in Section 3.

At the motor control level, special care is taken to ensure smooth movements. The PID controller boards ensure that the joints follow the desired trajectories. It is not desirable to follow the trajectory closely because this results in stiff movements and walking instability. The PID controller boards are programmed to ensure that the motors reach the desired joint position using small acceleration and high speed. This generates smooth and springy movements.



## **2 Identification of effective control variables**

Before starting any learning it is crucial to identify the effective control variables as well as their reasonable ranges. Inappropriate control variables can prevent the robot from learning. Many of these variables can have conflicting effects and since there is no precise knowledge about their interaction, it is necessary to minimize their use, as much as possible, in order to avoid conflict. In the case of biped walking, supporting ankle joint and foot positions are the most important control variables.

### **2.1 Ankle joint control**

Supporting ankle joints are most effective when the robot's center of gravity is above the joint axis. Ankle torque cannot effectively resist the overwhelming momentum caused by the body's deviation from the vertical. When the robot is in a vertical position then very small deviations of the ankle can have a significant effect on stability.

A reasonable range of variation of ankle joint position can be identified by putting the robot in a standing posture and slowly changing the joint position. A position magnitude that has a clear visual effect on the posture should be used as the maximum allowable value. A reasonable joint rate of change can be identified in the same manner.

Once the range of variation has been established, then the robot should always keep the desired joint position of the supporting ankle within the defined range, relative to the horizontal. Keeping joint actions within a reasonable range provides proper control for vertical positions while it cannot harm stability otherwise. Notice, however, that if ankle joint position functions as a positive feedback then even small variations can throw the robot out of balance regardless of the posture.

### **2.2 Hip joint control**

Foot step position is the most effective control variable to correct large posture deviations. The biped can stop a fall by putting the free foot appropriately on the ground at the right time. This would be the only posture control variable for a robot that walks on stilts, or one that has free passive ankle joints.

Using foot positions requires computing the inverse kinematics of the biped in order to issue the appropriate joint position commands. In order to save computation time, hip joint positions are used instead. This substitution is valid in this application because foot positions are also determined by other constraints, namely obstacle and body height constraints. It is not an equivalent solution but it works reasonably well.

### 2.3 Knee joint control

The knee joint of the supporting leg is used to control the body height. Since the body height is closely related to step length, walking speed and total energy, it is difficult to determine an optimal target for the controller. This target is thus learned by the robot.

Knee joint control is not always effective because there are many situations where the motor is not strong enough to keep the body at a desired height. The controller must then learn to take this into account, and set reasonable knee joint targets.

## 3 Action generation

The learning architecture used here is heavily based on using random actions. Each action is rewarded according to the outcome it yields. The biped, however, does not only react to each individual action separately, but because of its complex dynamics it reacts to most of the frequency components of the action signal: a series of successive actions. These frequency components are not explicitly taken into consideration in the learning process. The individual actions are thus rewarded for a biped behavior for which they are not totally responsible. Despite the fact that TD learning can, in many situations, deal with this problem, the learning can be extremely slow if not impossible.

The frequency component problem is dealt with here by using low pass filters and small standard deviations. This seriously restricts the system's ability to search the action domain. By being able to search only narrow areas, the system becomes vulnerable to local minimum problems. Furthermore because of the small magnitude of the actions, the learning system will be unable to learn because no action can restore the biped's stability.

This problem can be solved by using relative random actions instead of absolute actions. Relative actions are generated by the learning system and have small magnitude. The action that is issued to the biped, however, is generated as follows:

$$a(t) = \lambda a(t-1) + (1-\lambda) \sum_{i=0}^{t-1} n(i) + n(t) \quad (48)$$

where  $a(t)$  is the action at time  $t$ ,  $n(t)$  is the random relative action and  $\lambda$  is a decay factor. Assuming that the control signal is relatively smooth, the difference between successive actions is then very small. This assumption is valid in the case of biped walking because fast changing action signals easily lead to instability. This action generation method allows the learning system to use small random relative actions while providing the biped with the necessary action magnitude. There are still many undesired frequency components in the action signal, but they have a very small magnitude and they do not

contribute to the absolute action. This dramatically reduces their effect on the biped's dynamics. The decay factor allows the system to slowly forget past actions.

The learning algorithm still has a reward problem because the biped reacts to the absolute action while it learns from the relative actions. This can be easily solved with TD learning, because it is strictly a problem of delayed rewards. The difference between this problem and the problem of frequency components is that in this case the system's action is directly related to past individual actions.

### 3.1 Bang-Bang control

The action generation method described previously can also be used to apply bang-bang control to the case of continuous action. Using a small sampling period, and small bang-bang action magnitude as well as a low pass filter, can generate a continuous action signal. Figure 32 shows an example where the desired action is a sine wave. The graph shows the bang-bang action without low pass filtering.

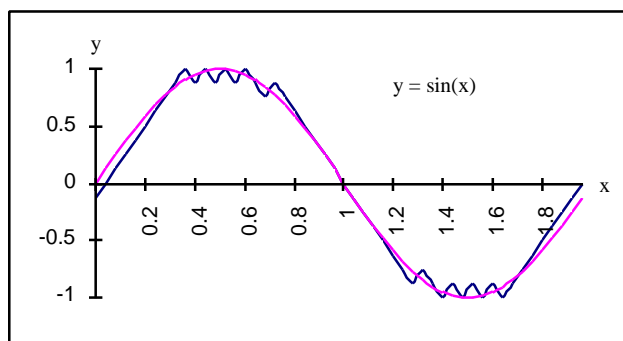


Figure 32: Continuous action using bang-bang control

Notice that if the sampling period is much higher than the rate of change of the optimal continuous action, and if the bang-bang relative action is randomly generated using an optimal probability distribution, then in theory and with appropriate low pass filtering, we can obtain an exact fit to the optimal continuous action. The bang-bang action acts then as a stochastic pulse width modulation signal.

This action generation method is used here only on the case of large perturbation where accuracy is not a major concern, and where the input space can have very low resolution. This is especially used when the body posture deviation from the vertical is very large. Because this method uses discrete action and low resolution inputs the learning is very fast. This method is not accurate enough to use with small perturbations or on nominal behavior.

## 4 CMAC pre-training

### 4.1 CPG CMAC pre-training

The CPG actor CMAC is pre-trained before using either the simulator or the biped. It is pre-trained using reasonable walking patterns. These walking patterns assume constant body height and constant walking speed. Desired joint positions are generated using simple foot positions. The  $x$  position of each foot changes linearly back and forth, and the  $y$  position is zero during half of the walking period and goes up then down during the other half. Right and left foot positions are always in quadrature.

Figure 33 shows desired foot positions where  $T$  is the walking period,  $S$  is the step length and  $H$  is the step height.

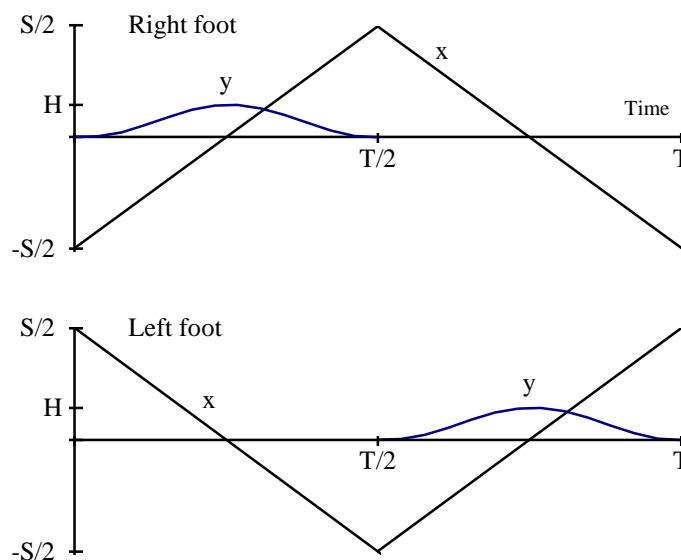


Figure 33: Pre-training foot positions

These foot positions are generated as follows:

for  $0 < t < T/2$ :

for  $T/2 < t < T$ :

$$\begin{aligned}
x_r &= -\frac{2S}{T}(t - \frac{T}{2}) + \frac{S}{2} & x_r &= \frac{2S}{T}t - \frac{S}{2} \\
y_r &= 0 & y_r &= \frac{H}{2}(\sin(\frac{4\pi}{T}t - \frac{\pi}{2}) + 1) \\
x_l &= -x_r & x_l &= -x_r \\
y_l &= \frac{H}{2}(\sin(\frac{4\pi}{T}(t - \frac{T}{2}) - \frac{\pi}{2}) + 1) & y_l &= 0
\end{aligned} \tag{49}$$

The constant body height is chosen so that the leg is completely extended at the end of each step. This can be easily determined knowing the step length and the extended leg total length. Desired joint positions can be determined using inverse kinematics. The step maximum height  $H$  can be either constant or proportional to the step length.

The CPG CMAC is trained with these joint positions using different step length and walking period values. Simple LMS supervised learning is used in this case. The walking period  $T$  ranges from 0 to 2 seconds, the step length  $S$  ranges from .1m to .3m. The time  $t$  is periodic with period  $T$ .

The CMAC has 4 generalization layers and a resolution of 10 for each of the three input variables, time, walking period, and step length. An input resolution of 10 means that the CMAC can read 10 distinct values for that input.

#### 4.2 Body height CMAC pre-training

The body height CMAC is trained using a constant target height. The actual height value is generated in the same manner as in the case of CPG training. If  $L$  is the total length of the extended leg and  $S$  is the step length, then the body height  $h$  is computed as follows:

$$h = \sqrt{L^2 - \left(\frac{S}{2}\right)^2} \tag{50}$$

The body height CMAC uses four generalization layers and uses two inputs: step length and body height. The input resolution is 10 for each input. The body height input is not taken into consideration during the pre-training phase. It is more relevant during real walking experiments where the CMAC has to learn globally efficient body height trajectories.

The leg length  $L$  here does not include the foot because the latter is usually horizontal. In this case the computed body height should be incremented by the foot height.

### 4.3 Posture CMAC pre-training

The posture control CMAC is pre-trained by two parallel posture teaching controllers using the biped simulator. Each teaching controller monitors the posture and issues a PD control action. The first controller's action is added to the Posture CMAC ankle joint output. The second controller's action is added to the CPG's step length command thus controlling foot positions.

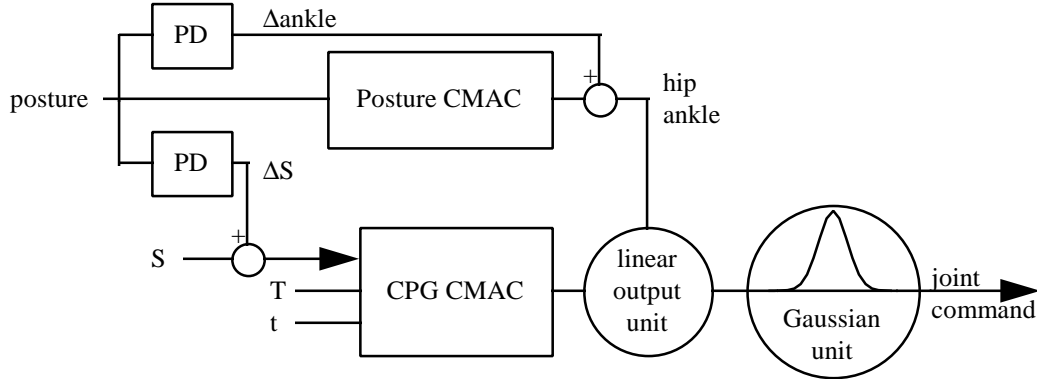


Figure 34: Posture CMAC pre-training

The posture CMAC learns from the PD controller because the random action generated by the Gaussian unit is directly related to the teaching controllers' actions. The PD controllers can effectively control the posture only when the biped has a linear behavior. The role of this teacher is to bring the CMAC's weights to the neighborhood of an optimal solution.

Figure 34 shows the posture CMAC teaching architecture. The second PD controller seems to be used in series rather than in parallel, but in effect it is used in parallel. The figure can be modified by splitting the CPG into two parallel controllers; one would generate joint position regardless of the posture, and the other would be in series with the PD controller.

The posture CMAC inputs are three successive posture readings:  $p(t)$ ,  $p(t-1)$ , and  $p(t-2)$ . It uses four generalization layers. The input resolution is 20 for  $p(t)$ , 20 for  $p(t-1)$  and 5 for  $p(t-2)$ . The inputs are truncated at  $\pm\pi/5$  because it is very difficult for the robot to recover its stability after such a large deviation from the vertical.

The standard deviation of the Gaussian is zero during the pre-training period.

#### 4.4 Critic pre-training

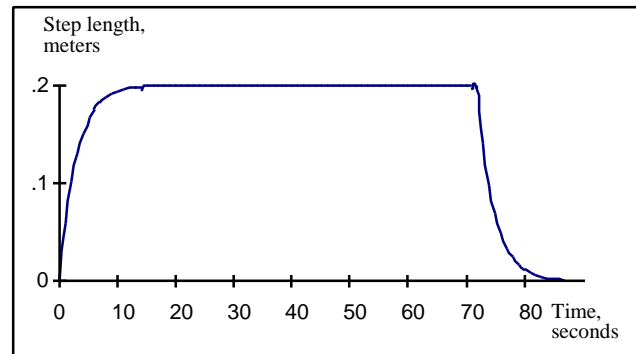
The critics are pre-trained by using a non-zero standard deviation. This allows them to learn general evaluations of each state, and locate failure zones and danger areas. The standard deviation starts at 0, then it changes at a random time in order to be able to scan as many states as possible.

### 5 The Simulator

The simulator is mainly used to pre-train all of the CMAC neural networks in order to speed up learning. Even though the simulator is supposed to be equivalent to the hardware robot, there are many differences that cannot be reconciled. A simulator solution works reasonably well in normal walking conditions. In case of large perturbations, however, the differences become predominant and the solution is not sufficient anymore.

Simulator experiments consist of different phases of pre-training. Some of the CMACs are trained in individual experiments and some are pre-trained during normal walking experiments where the whole learning architecture is involved.

Each walking experiment starts with zero step length and increases until it reaches the desired value. Before stopping, the step length decreases continuously until it reaches zero. Figure 35 shows a typical example of step length command over time. This is necessary because the learning architecture is strongly dependent on continuity. Any abrupt change in the step length can cause instability.



*Figure 35: Step length command*

One of the most important reasons why the simulator is used is the ability to recover from failures. When the robot fails it is reset to its starting position and the learning continues.

The robot always starts standing up in a straight posture. This is a stable position from which the biped can move continuously towards walking. Standing up can be considered as a walking pattern with zero step length.

A failure signal is generated every time the posture exceeds  $\pm \pi/5$ , or the body height gets below .6m. There is no failure related to the step or energy constraints because a significant violation of these constraints usually leads to a failure in posture or height. A negative reinforcement is, however, always generated for every constraint that gets violated.

The posture CMAC is activated when the posture exceeds  $\pm \pi/20$ , and the height CMAC is activated when the difference between the body height and the target height exceeds  $\pm 2$ cm. The target height is determined by computing the running average of the height. An optimum height is reached once the CPG has learned.

The energy constraint continuously issues a reinforcement proportional to the negative of the total energy. The obstacle constraint issues a reinforcement equal to -1 when the biped hits an obstacle and 0 otherwise.

Individual controller action magnitudes are limited in range in order to minimize conflict between different controllers. Also joint position commands are continuously checked to ensure that they are reasonable.

Sometimes the simulator can have abnormal behavior because it uses a discrete time, while system dynamics are a function of continuous time in reality. In order to minimize this kind of problems, smooth trajectories should be used and only reasonable joint and foot positions can be allowed.

## **6 The Biped**

### **6.1 stability issues**

In order to conduct learning experiments on the hardware biped, it is necessary to improve mechanical stability. Due to mechanical instability the biped may never learn, because its dynamics will be extremely complex, if not chaotic. Smooth movements alone are not sufficient to ensure this stability, an appropriate weight distribution can dramatically increase mechanical stability.

The walking cart must be as light as possible in order to minimize its interference with the biped's dynamics. It must however be hard to turn and lift off the ground, otherwise it will not perform its primary function which is to keep the robot within the sagittal plane. Placing weights at the extremities of the cart increases the moment necessary to lift its



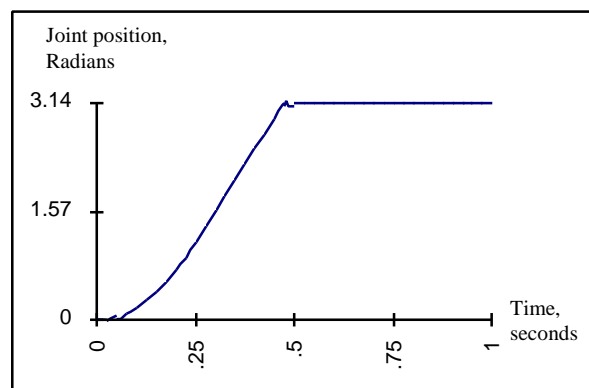
wheels off the ground or rotate around the biped. Fortunately these weights do not need to be large thus the total cart weight is kept relatively low.

The body moment of inertia must be large compared to the legs. Indeed, because of the moment conservation law, the body must rotate as the legs move. This movement magnitude is inversely proportional to the inertia. In the extreme case where the body has very small inertia compared to the legs, the leg will be still while the body will be oscillating following the hip joint movements. The desired behavior is the other way around. An ideal biped will have mass-less legs and a heavy body.

A compromise must be found between having a body heavy enough to oppose leg movements, and light enough so that the motors can hold it. A judicious placement of weights on the body can substantially improve stability. Placing the weights near the hip joints and on the sides increases the moments that oppose rotation around both vertical and horizontal axes.

Joint level control uses PD controllers. As mentioned earlier a stiff control is undesirable because it can introduce instability. A smooth and springy control is much more suitable for stability and produces natural looking movements. Integral control can also be very detrimental to stability. It generates abrupt movements, because it keeps on accumulating position error until it reaches a level strong enough to drive the motor, then the motor speed suddenly increases and causes a major perturbation.

The PD controllers used on the biped can be programmed to control the maximum acceleration and speed that the motor uses to reach the desired position. Using small acceleration ensures smooth movements and large speed ensures trajectory following. Figure 36 shows a typical step response curve. Notice that the response time is almost proportional to the step magnitude. Since the PD controllers are given step commands of small magnitude, the response time is reasonably small.



*Figure 36: Step response*

It is possible to obtain smooth trajectories using small PD gains instead of limiting acceleration and speed. This, however, would seriously diminish motor torque and limit biped controllability.

When the biped is subject to too many perturbations, it cannot deal with them efficiently while keeping its stability. To help it recover, the desired walking speed diminishes as the perturbations accumulate. At low walking speed the biped has a much better chance to regain stability than at high speed. The walking speed command diminishes proportionally to the running average of the absolute value of the posture as follows:

$$\begin{aligned}\Delta S(t) &= ke(t) \\ e(t) &= \lambda e(t-1) + (1-\lambda)|p(t)|\end{aligned}\tag{51}$$

where  $e(t)$  is reset to 0 after each failure.

## 6.2 Falling safeguard

The biped has no real safeguard against falling over. The only control used to delay the fall and minimize its impact is to extend both legs as in a standing position. This control succeeds many times in stabilizing the robot especially when the perturbations are not too large.

This safeguard has no effect on learning, it is only used to protect the biped's health.

## 6.3 Extra learning with bang-bang control

Because of the differences between the simulator and the biped, simulator walking solutions do not work efficiently on the biped. They are sufficient in many cases but cannot be considered optimal for the hardware biped. It is thus necessary to use learning on the biped in order to optimize these solutions. In the case of small perturbations, simulator solutions achieved reasonable walking. With large perturbation, however, the biped could not keep its balance.

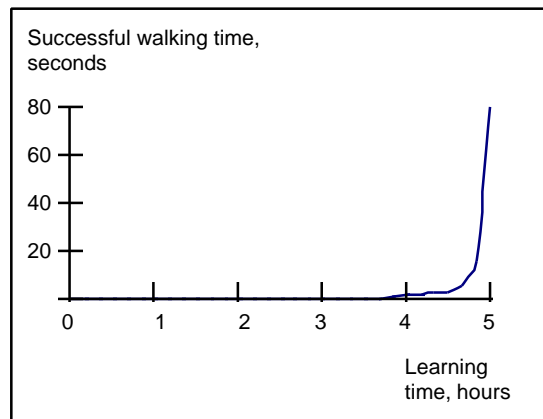
It would take a long time to train the CMACs using continuous action in real-time. Discrete action can learn much faster and it is appropriate for large perturbation control. As described previously a bang-bang relative action is used to generate continuous absolute actions. This bang-bang controller has the same inputs and outputs as the posture CMAC and is used in parallel. It uses a one-layer CMAC with low resolution inputs (5 for each input). The bang-bang action is activated only when the posture deviation exceeds  $\pm \pi/10$ .

This extra controller helps achieving stable dynamic walking on the biped, and in the long term, it increases learning speed for the other CMACs by working as a teacher.

## 7 Results

### 7.1 Simulator learning curve

The learning curve in Figure 37 shows the average successful walking time. The walker learns to walk after 5 hours approximately. The experiment is then stopped after a walk of 80 seconds, but the walker can in general walk indefinitely without falling over. Since the simulator does not run in real time, the experiment's computation time using a 486/66Mhz PC is less than 3 hours. If such an experiment were to be conducted on the hardware biped, it would take more than 15 hours because of the time it takes to restart the biped after each failure.



*Figure 37: CPG learning curve*

Before this experiment only the CPG is pre-trained with reasonable joint positions. The pre-training time is very small (15 min.) because it is a simple supervised learning problem.

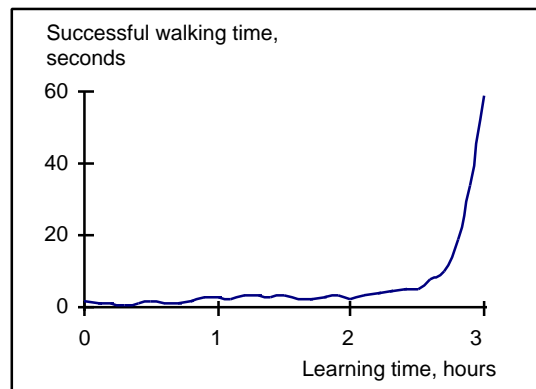
### 7.2 Hardware biped learning curve

The hardware biped cannot walk successfully using the simulator pre-trained CMACs only. The average walk time is less than 3 seconds. However, it has the right kind of joint movements, and with slight intervention from the user when the posture is too large, it can walk reasonably well. Extra learning is thus necessary and promising.

The extra learning is mostly done by adding a posture controller that uses bang-bang control. This controller intervenes at the hip joint level and provides the extra help that the robot needs in order to walk. Since this controller is only needed for large perturbations it does not require high input resolution. It learns much faster than a continuous action controller with high input resolution.

The CPG quickly adapts to the hardware biped because it learns from the bang-bang controller's actions. Because the latter also provides a safeguard, it gives the CPG the chance to learn without constantly failing.

Figure 38 shows the hardware biped learning curve. It learns after approximately 3 hours of walking time. The dead time caused by failure adds about 3 more hours to the experiment.



*Figure 38: Extra learning curve*

### **7.3 Foot positions**

The following foot position figures show a typical walk of the hardware biped. The step length is about 10 cm and the walking period is about 1.2 seconds. The maximum step height is about 3 cm.

Figure 39 and Figure 40 show the right and left foot positions. These are similar to the pre-training foot positions shown in Figure 33. Notice that the x foot trajectory averages are less than zero. This means that the robot walks with its feet slightly lagging behind. This behavior can be explained by the cart that the biped is pushing. Since the cart is relatively heavy, the biped needs to lean forward in order to increase its pushing force. Humans behave the same way when they need to push something too heavy.

Notice also that when the biped lifts a leg, it does not immediately move it forward. When it puts the leg on the ground, however, it immediately moves it backwards. The latter behavior is obvious because moving the supporting leg forward would cause instability. One plausible explanation of the first behavior is that the robot uses the free leg as a counter-balance to prevent it from falling over forward, and to absorb the energy caused by the foot impact on the floor.

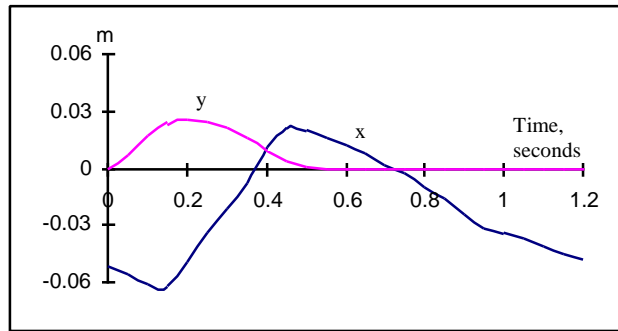


Figure 39: Real foot positions:  $x_r$ ,  $y_r$

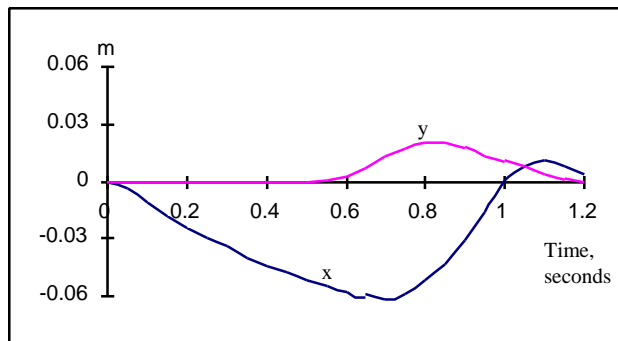


Figure 40: Real foot positions:  $x_l$ ,  $y_l$

Figure 41 shows the right and left foot x trajectories. Forward movement corresponds to the rising portion of the trajectory and backward movement corresponds to the falling portion. Notice that the biped moves the legs forward much faster than backwards. One reason for this behavior is that the supporting leg is subject to large pressure, thus it cannot move as fast as the free leg. Another plausible reason is that quick movements of the supporting leg can cause large instability.

Notice that at the beginning of every step, both feet move backwards. This is probably a safe way to deal with the problem of double support phase. Indeed if there is a double

support phase and the feet try to move in opposite directions, then the biped will become unstable as soon as one foot gets off the ground.

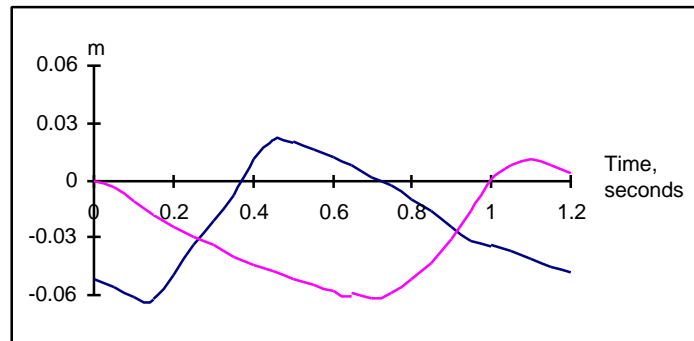


Figure 41: Real foot positions:  $x_r$ ,  $x_l$

Figure 42 shows the right and left  $y$  trajectories. They do not have the same magnitude because the biped is not perfectly symmetrical. The asymmetry may also be due to the composition of the walking cart: it is made of very thin aluminum bars that bend easily. Because of this the biped seems to be slightly limping in its walk.

The overlap between the right and left foot trajectories is minimal. This alleviates the problem of closed chain kinematics and simplifies robot dynamics.

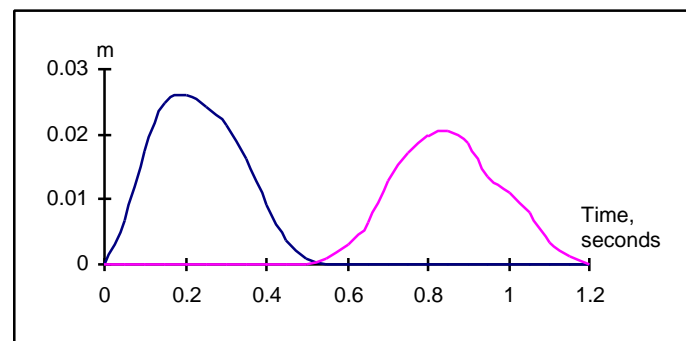
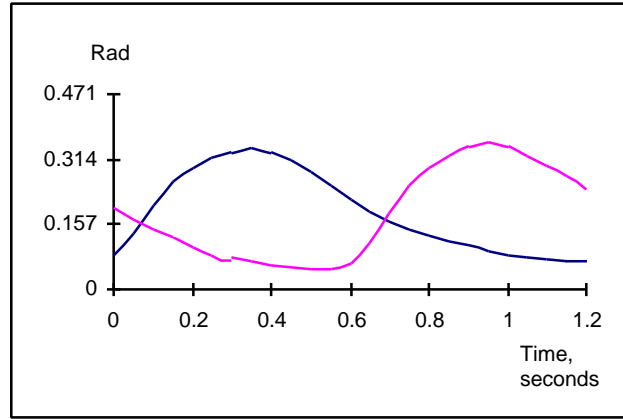


Figure 42: Real foot positions:  $y_r$ ,  $y_l$

## 7.4 Real joint positions

Figure 43 shows the right and left hip joint trajectories. The increasing portion of the curve corresponds to moving the leg forward, and the decreasing portion corresponds to

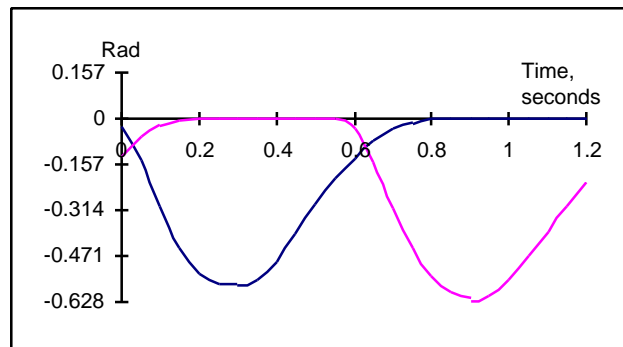
moving the leg backwards. Notice that the averages of these trajectories are greater than zero. This means that the biped is slightly leaning forward. The reason for this behavior can be the same as for the case of lagging foot positions: the robot needs to lean forward in order to be able to push the walking cart.



*Figure 43: Real joint positions: hips*

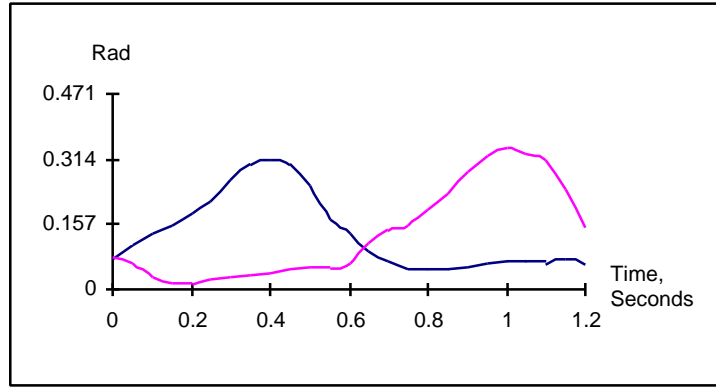
Figure 44 shows the right and left knee joint trajectories. The supporting leg knee joint is most of the time equal to zero, because the knee joint is mechanically restricted. This provides an energy efficient walking pattern. The biped's weight is supported by the mechanical stops of the knee joint instead of the DC motor torque.

Actually there is no other alternative to locking the supporting knee because the DC motor is not strong enough to hold the biped's weight.



*Figure 44: Real joint positions: knees*

Figure 45 shows the right and left ankle joint trajectories. The ankles make sure the feet are constantly flat while slightly resisting to posture disturbances. Notice that the averages of these trajectories are greater than zero. This is due to the biped slightly leaning forward.



*Figure 45: Real joint positions: ankles*

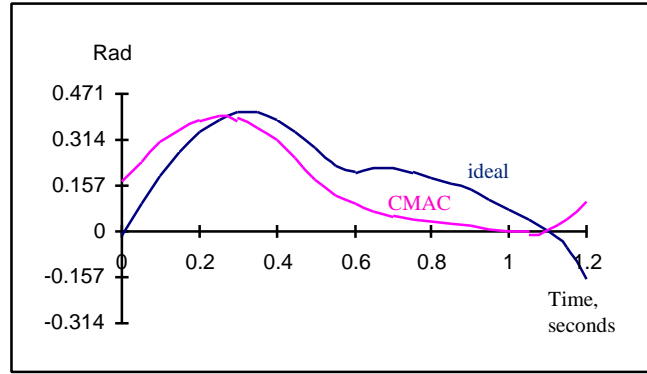
## 7.5 CPG CMAC learning

The following graphs show the ideal joint trajectories that were used to pre-train the CPG CMAC, versus the CPG learned trajectories. The learned trajectories are much smoother than the ideal ones. This is due to the CMAC's relatively low input resolution. Another reason is that the CPG learns a solution that satisfies most of the constraints. This yields an averaging behavior, thus smooth trajectories.

The CPG learned trajectories shown here are the final CPG actions which were learned after the walking experiment. They are not the CMAC trajectories that were learned before the experiment. They are issued as commands to the DC motor PD controllers.

Only right leg joints are shown because left leg joints are similar.



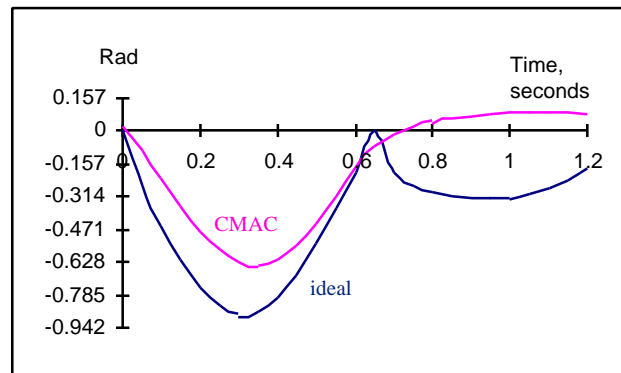


*Figure 46: Ideal vs CPG joint positions: hip*

Figure 46 shows the hip joint ideal versus learned trajectories. Notice that the learned trajectory is much smoother than the ideal one.

Figure 47 shows the knee joint ideal versus learned trajectories. Notice that in the learned trajectory the knee joint command becomes positive during the support phase (between .6sec. and 1.2sec.). Because of the mechanical stop on the knees the real joint position can never be positive. Issuing a positive command ensures that the knee is tightly locked.

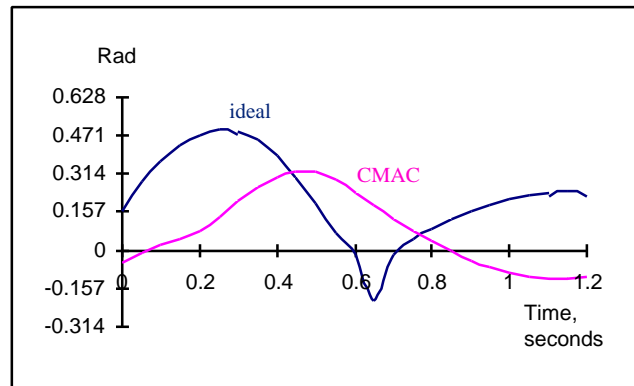
The large difference between the two curves in the support phase can be explained by the fact that ideal joint positions are generated assuming constant height. By locking the knees, the biped cannot keep constant height.



*Figure 47: Ideal vs CPG joint positions: knees*

Figure 48 shows the ankle joint ideal versus learned trajectories. In the ideal case the foot is supposed to stay flat at all times. This is equivalent to having a free passive ankle or

walking on stilts. In the learned trajectory the foot is supposed to stay flat, while exerting slight pressures to attenuate posture perturbations. It is also supposed to change smoothly.

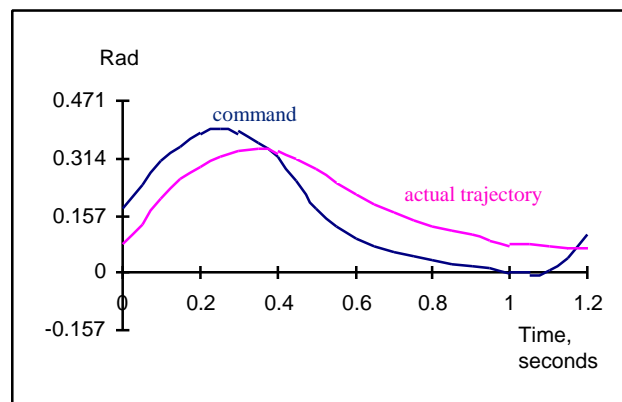


*Figure 48: Ideal vs CPG joint positions: ankles*

## 7.6 Joint level control graphs

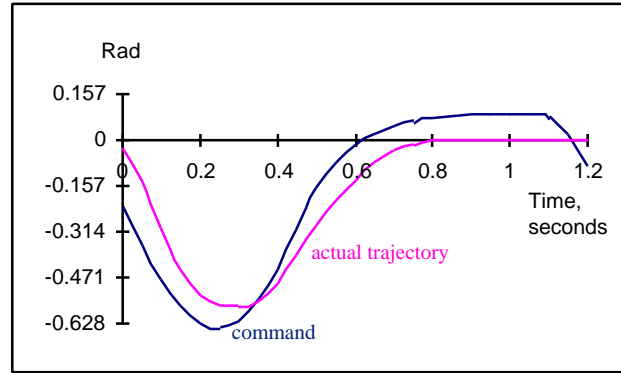
The following graphs show PD control command versus real joint trajectories. Only right leg trajectories are shown.

Figure 49 shows hip joint command and real trajectories. The real trajectory is lagging behind the command because of the smoothing that was programmed in the PD controller. As discussed earlier, a tight control would induce instability and rough movements.



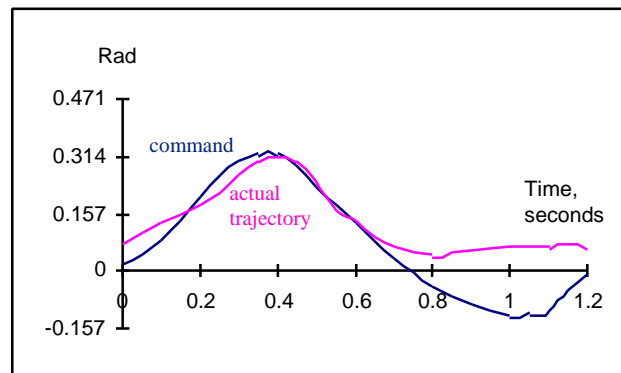
*Figure 49: Hip joint control*

Figure 50 shows knee joint command and real trajectories. Notice that the real joint cannot be positive because of the mechanical knee stop. The real trajectory is lagging in the same manner as the hip trajectory.



*Figure 50: Knee joint control*

Figure 51 shows ankle joint command and real trajectories. In this case the delay is not caused only by the PD controller programming. It is mostly due to the DC motor which is not powerful enough to follow the desired trajectory.

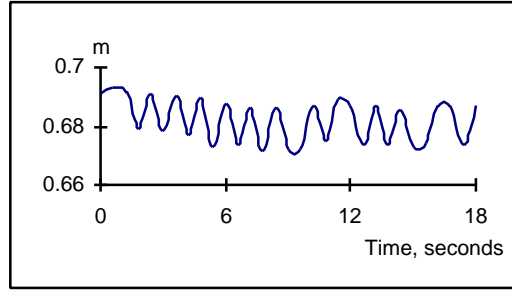


*Figure 51: Ankle joint control*

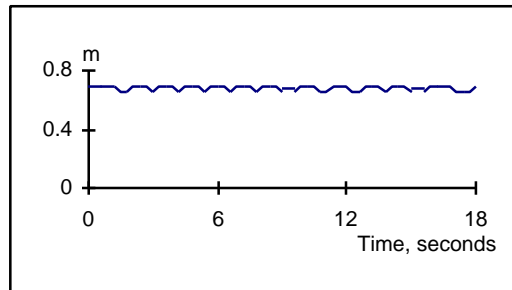
From a pure control point of view, these trajectories show very poor performance. In this case, however, they show great learning performance because this means that the learning algorithm could find a solution that takes into account the limitations of the DC motors and PD controllers. The delays and smoothing processes are considered by the learning algorithm as part of the system's dynamics.

## 7.7 Body height

Figure 52 and Figure 53 show the biped's body height over an 18 second walk. The body height is not constant because the supporting knee is constantly locked. The variation is, however, not significant (Figure 53). Notice the slow start due to the step length command which starts at zero, and increases until it reaches the desired step. This usually takes about 2 seconds.



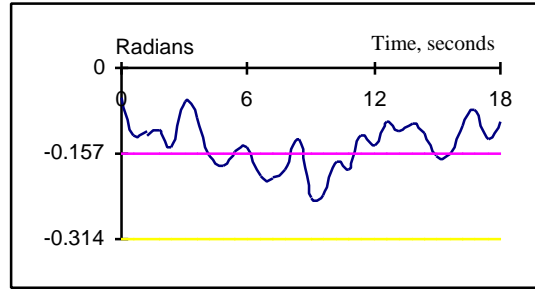
*Figure 52: Body height*



*Figure 53: Body height*

## 7.8 Body posture

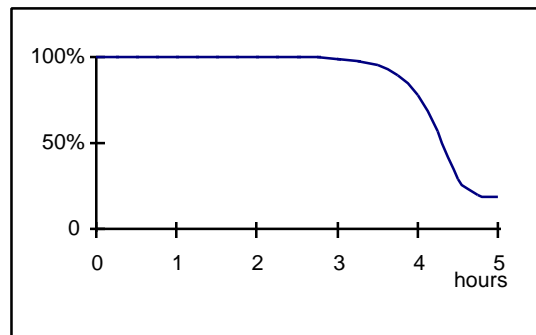
Figure 54 shows the biped's body posture over an 18 second walk. Notice that the posture is slightly negative. This means that the biped is leaning forward, for the reasons stated previously. The  $-\pi/20$  line indicates the threshold when the posture peripheral controller intervenes.



*Figure 54: Body posture*

## 7.9 Posture intervention

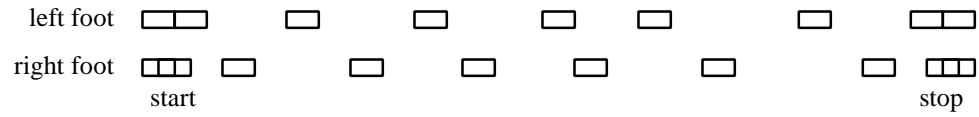
Figure 55 shows the frequency at which the posture constraint is violated during the learning experiment. Before the CPG has learned an appropriate pattern, the posture controller intervenes almost 100% of the time. Once most of the learning has been achieved, the posture constraint is violated only about 15% of the time. This intervention ratio cannot be null in reality, because this would mean that an open loop solution to biped dynamic walking has been found, which is very unlikely.



*Figure 55: Posture constraint intervention frequency*

## 7.10 Foot patterns on the floor

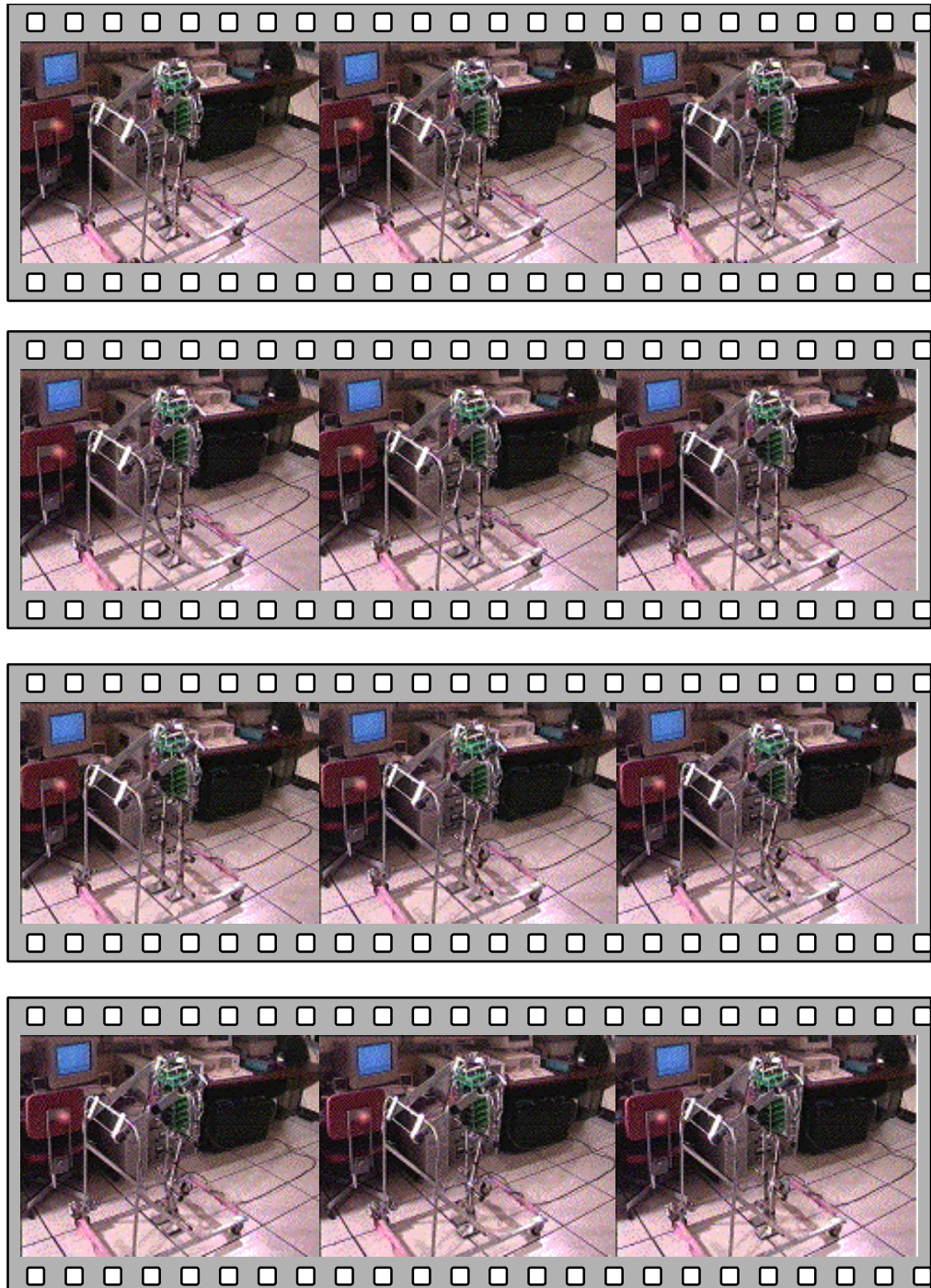
Figure 56 shows a typical foot placement pattern. The biped starts and ends the walking with zero step length. Notice that the step length changes many times during a walk. This is mainly caused by the posture controller which moves the hip joints in a direction that opposes posture deviations. When the biped is falling forward, for instance, the step length increases in order to stop the fall, and if it is falling backwards, then the step length shortens in order to allow the biped to take advantage of its inertia that generally pushes it forward.



*Figure 56: Typical foot placements*

## 7.11 Biped movie frames

Figure 57 shows a frame by frame movie of the biped taking two steps, a right foot step then a left foot step. It starts by lifting the right leg, moving it forward, and putting the right foot on the ground. During this step the left foot is on the ground and moving backwards. During the second step the biped lifts the left leg, moves it forward, and puts the left foot on the ground; while the right foot is on the ground and moving backwards.



*Figure 57: Biped movie frames*

## CHAPTER SEVEN

### CONCLUSION

During this study a hardware biped robot was built and a new reinforcement learning algorithm (SSR) as well as a new learning architecture were developed. The biped learned dynamic walking without any previous knowledge about its dynamic model. The learning and control were done using relatively small computing power.

The biped contains all of the necessary motor control electronics, as well as a local processor that communicates with the main computer using RS232. This design allows the main computer to deal only with the high level commands, does not require any extra hardware, and can work with different platforms and different operating systems. The biped uses low gear ratio DC motors in order to provide smooth and flexible movements. One of the implications of this is that the legs can move freely when no power is supplied. This was done in order to achieve movements as close to human movements as possible.

The Self Scaling Reinforcement learning algorithm (SSR) was developed to deal with the problem of reinforcement learning in continuous action domains. It allows fast learning and avoids overshooting, which is a typical problem in continuous action domains. It also adjusts the range of action domain search according to the actual performance. The better the performance the less the system needs to explore the action domain.

The learning architecture succeeded in dealing with the problems of large numbers of inputs, knowledge integration and task definition. It consists of a central controller and several peripheral controllers. Because of its modular nature, it is possible to use several neural networks with small numbers of inputs instead of one large neural network. This dramatically increases the learning speed and reduces the demand on memory and computing power. The architecture also allows easy incorporation of any knowledge by adding a peripheral controller that represents that knowledge. The walking performance immediately benefits from this extra knowledge and the other controllers learn from it. The walking task can also be shaped by adding extra peripheral controllers that constrain the walking to respect the new requirements, i.e. desired posture, body height, etc.

Some of the CMAC neural networks used to control the biped have been pre-trained in order to increase the learning speed. After the pre-training, the system was free to learn new solutions. These solutions were not in any way constrained by the initial knowledge.



Indeed, the results show that the learned trajectories can be in many cases very different from those obtained with pre-training. In many cases learning methods can not be used in real-time applications because of the long learning time they require and because many systems can not allow for major failures. Pre-training can thus make learning methods more practical for real-time control. There are, however, situations where pre-training can be harmful. One can imagine a pre-training process that leads the system to a local minimum that does not represent a viable solution. The system might not be able to get out of that local minimum, thus never learn.

The learning algorithm uses many parameters that have significant effects on learning and adaptation speed, immunity to noise, etc. Unfortunately there is no explicit way of determining the optimal values of these parameters. Most of the values are determined by trial and error.

Learning rates are in general chosen to have a value of .9 approximately. The SSR algorithm is designed to work best with a learning rate equal to 1. The learning rate has a direct effect on learning speed and immunity to noise. A large learning rate would make the system vulnerable to overshooting and to noise. A small learning rate provides excellent immunity to noise, however it seriously decreases the learning speed.

Decay factors determine the amount of past history that has a direct effect on the system's future performance. A decay rate that is too small would prevent the system from learning because it would deprive it from valuable past information. A decay rate that is too large would provide the system with too much information. The learning algorithm would then waste time filtering out the unnecessary information.

One more factor that has a critical impact on learning is the SSR factor that controls the action domain search. The size of the search domain is determined by the standard deviation of the Gaussian unit. If the standard deviation is too small, the system will have a very small search domain. This decreases the learning speed and increases the system's vulnerability to the local minima problem. If the factor is too large, the system's performance will not reach its maximum because there will always be a randomness even if the system has learned an optimal solution. It is in general safer to use a large factor than a small one.

Even though the learning algorithms and architecture used here have successfully solved the problem of dynamic biped walking, there are many improvements that can be added to increase learning speed, robustness, and versatility. One major improvement would be to intelligently determine the extent of the contributions of each peripheral controller based on their individual performance and expertise. Another improvement is to find a way of determining optimal values for different learning parameters as described above. Actually most learning methods are in dire need of such an improvement. The performance may also be improved by dynamically setting the PID gains to deal with each specific situation.

The learning architecture can also be improved by adding a special controller that ensures that the biped is always in a state where the deviations from the nominal behavior are not too great. This controller does not need to be accurate since the learning architecture can control the biped near nominal states. The peripheral controllers currently do achieve some of this functionality, however, it would be possible to increase the range of allowed perturbations.

While these improvements can be beneficial, it is very important to achieve them using systems of realistic sizes and complexity. One of the main objectives of this research is to solve the problem of biped walking using the simplest means possible. Indeed all of the computing power used here can reside within the body of the actual biped. In the near future, it will be easier to get fast and large computing systems of very small physical size. Some of the compromises made in this research will then not be necessary.

There are many things that can be learned about dynamic biped walking from the solutions provided by the learning system. Joint trajectories especially show a certain behavior that reflects many interesting characteristics of biped walking. Some of these characteristics are the choice of locking the knee of the supporting leg, and moving both feet backwards to avoid double support phase problems. A thorough observation of the biped's behavior may reveal many more things that can be taken advantage of in a more sophisticated controller, or implemented as peripheral controllers in the present architecture. More things can also be learned by subjecting the biped to different situations and observing how it deals with them. Evidently some of these learned things represent optimal behavior, and others are there because of the nature of the biped and the learning architecture. A careful selection is thus necessary in order to avoid forcing the biped to choose a behavior that is not necessarily optimal.

This research has achieved one of its most important goals which is to solve the problem of biped dynamic walking with no prior knowledge of the biped's dynamic model. This was achieved using reinforcement learning and a modular learning architecture. This is another proof that learning methods are able to solve complex problems, even if there is no absolute proof of convergence or stability for many learning algorithms.

Learning methods will become more powerful as a solid theory is developed to support them, and a deeper understanding of animal learning is reached. Hopefully it will be possible to mimic the brain and use the theory to make it perfect. After all, if science gave man actuators stronger than his own muscles, made him fly, and see in the dark; then it is time it made him think a little more.

## REFERENCES

- J. S. Albus, "A new approach to manipulator control: the cerebellar model articulation controller (CMAC)," *ASME Journal of Dynamic systems, Measurements, and Control*, pp. 220-227, September 1975.
- J. S. Albus, "Data storage in the cerebellar model articulation controller," *ASME Journal of Dynamic systems, Measurements, and Control*, pp. 228-233, September 1975.
- A. G. Barto, R. S. Sutton, C. W. Anderson, "Neurolike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, pp. 834-846, September/October 1983.
- A. G. Barto, S. P. Singh, "Reinforcement learning and dynamic programming," *Proceedings of the 6th Yale Workshop on Adaptive and Learning Systems*, 1990.
- J. S. Bay and H. Hemami, "Modeling of a neural pattern generator with coupled nonlinear oscillators," *IEEE Transactions on Biomedical Engineering*, vol. BME-34, no. 4, April 1987.
- H. Benbrahim, J. S. Doleac, J. A. Franklin, O. G. Selfridge, "Real-Time learning: a ball on a beam," *Proceedings of the 1992 International Joint Conference on Neural Networks (IJCNN)*, vol. 1, pp. 98-103, June, Baltimore, MD.
- H. Benbrahim, J. A. Franklin, "Self-scaling reinforcement: a new algorithm for learning continuous control," *Proceedings of the 8th Yale Workshop on Adaptive and Learning Systems*, 1994.
- J. A. Franklin, "Refinement of robot motor skills through reinforcement learning," *Proceedings of the 27th IEEE Conference on Decision and Control*, December 1988.
- J. A. Franklin, "Historical perspective and state of the art in connectionist learning control," *Proceedings of the 28th IEEE Conference on Decision and Control*, December 1989.
- E. C. Freuder, R. J. Wallace, "Partial constraint satisfaction," *Artificial Intelligence*, vol. 58, pp. 21-70, 1992.
- R. E. Goddard, Y. F. Zheng, H. Hemami, "Control of the heel-off to toe-off motion of a dynamic biped gait," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 1, January/February 1992.
- S. Grillner, "Some aspects on the descending control of the spinal circuit's generating locomotor movements," in *Neural Control of Locomotion*, R. M. Herman *et al.*, Eds. New York: Plenum, 1976.
- V. Gullapalli, "A stochastic reinforcement learning algorithm for learning real-valued functions," *Neural Networks*, vol. 3, pp. 671-692, 1990.

- V. Gullapalli, J. A., Franklin, H. Benbrahim, "Acquiring robot skills via reinforcement learning," *IEEE Control Systems Magazine*, vol. 14, no. 1, pp. 13-24, February 1994.
- J. K. Hodgins, M. H. Raibert, "Biped Gymnastics," *The International Journal of Robotics Research*, vol. 9, no. 2, April 1990.
- R. A. Jacobs, M. I. Jordan, A. G. Barto, "Task decomposition through competition in a modular connectionist architecture: the what and where vision tasks," *Cognitive Science*, vol. 15, pp. 219-250, 1991.
- M. I. Jordan, "Constrained supervised learning," *Journal of Mathematical Psychology*, vol. 36, pp. 396-425, 1992.
- M. I. Jordan, D. E. Rumelhart, "Forward models: supervised learning with a distal teacher," *Cognitive Science*, vol. 16, pp. 307-354, 1992.
- S. Kajita, T. Yamaura, A. Kobayashi, "Dynamic walking control of a biped robot along a potential energy conserving orbit," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 4, August 1992.
- D. Katic, M. Vukobratovic, "Decomposed connectionist architecture for fast and robust learning of robot dynamics," *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, May 1992.
- I. Kato, S. Ohtheru, H. Kobayashi, K. Shirai and A. Uchiyama, "Information-power machine with senses and limbs," *First CISM-IFTOMM Symposium on Theory and Practice of Robots and Manipulators*, Springer-Verlag, 1974.
- S. Kawaji and K. Sawada, "Dynamical walking of biped locomotion robot with characteristic rhythm," *Japan/USA Symposium on Flexible Automation*, vol. 1, ASME 1992.
- S. Kitamura, Y. Kurematsu, M. Iwata, "Motion generation of a biped locomotive robot using an inverted pendulum model and neural networks," *Proceedings of the 29<sup>th</sup> Conference on Decision and Control*, pp. 3308-3312, December 1990.
- Y. Kurematsu, O. Katayama, M. Iwata, S. Kitamura, "Autonomous trajectory generation of a biped locomotive robot," *Proceedings of the 1991 International Joint Conference on Neural Networks (IJCNN)*, vol. 3, pp. 1983-8.
- P. W. Latham, "A simulation study of bipedal walking robots: modeling, walking algorithms, and neural network control," Ph.D. dissertation, University of New Hampshire, Durham, NH, September 1992.
- T. McGeer, "Passive dynamic walking," *The International Journal of Robotics Research*, vol. 9, no. 2, April 1990.
- D. Michie, R. Chambers, "Boxes: an experiment in adaptive control," *Machine Intelligence*, E. Dale, D. Michie, editors. Edinburgh, UK: Oliver and Boyd, 1968.
- W. T. Miller, F. H. Glanz, L. G. Kraft, "CMAC: an associative neural network alternative to backpropagation," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1561-1567, October 1990.

- W. T. Miller, R. P. Hewes, F. H. Glanz, L. G. Kraft, "Real-Time dynamic control of an industrial manipulator using a neural-network-based learning controller," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 1, February 1990.
- W. T. Miller, "Real-Time neural network control of a biped walking robot," *IEEE Control Systems Magazine*, vol. 14, no. 1, pp. 41-48, February 1994.
- H. Miura, I. Shimoyama, "Dynamic walk of a biped," *The International Journal of Robotics Research*, vol. 3, no. 2, pp. 60-74, Summer 1984.
- K. S. Narendra, M.A.L. Thathachar, "Learning Automata-a Survey," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-4, no. 4, pp. 323-334, July 1974.
- K. S. Narendra, J. Balakrishnan "Intelligent control using switching and tuning," *Proceedings of the 8th Yale Workshop on Adaptive and Learning Systems*, 1994.
- M. H. Raibert, "Hopping in legged systems—modeling and simulation for the two-dimensional one-legged case," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-14, no. 3, May/June 1984.
- D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning internal representations by error propagation," D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1: Foundations. Bradford Books/MIT Press, Cambridge, MA, 1986.
- A. Sano, J. Furusho, "Realization of natural dynamic walking using the angular momentum information," *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1476-81, 1990.
- M. Sato, K. Abe, and H. Takeda, "Learning control of finite Markov chains with explicit trade-off between estimation and control," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 18, pp. 677-684, 1988.
- C. Shih, Y. Zhu, W. Gruver, "Optimization of the biped robot trajectory," *Proceedings of the 1991 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 2, pp. 899-903, 1991.
- C. Shih, W. A. Gruver, "Control of a biped robot in the double-support phase," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 4, July/August 1992.
- J. Sklansky, "Learning systems for automatic control," *IEEE Transactions on Automatic Control*, vol. AC-11, pp. 6-19, 1966.
- S. Stitt, Y. F. Zheng, "Distal learning applied to biped robots," Y. F. Zheng editor. *Recent Trends in Mobile Robots*, World Scientific, 1993.
- R. S. Sutton, "Temporal Credit Assignment in Reinforcement Learning," Ph.D. thesis, University of Massachusetts, Amherst, MA, 1984.
- R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9-44, 1988.
- A. Takanishi, G. Naito, M. Ishida, I. Kato, "Realization of plane walking by the biped walking robot WL-10R," *Robotic and Manipulator Systems*, pp. 283-393, 1982.

- A. Takanishi, H. Lim, M. Tsuda, I. Kato, "Realization of dynamic biped walking stabilized by trunk motion on a sagittally uneven surface," *IEEE International Workshop on Intelligent Robots and Systems, IROS* '1990.
- B. Van Der Pol, "On relaxation-oscillations," *Philosophy Magazine*, ser. 7, vol. 2, pp. 978-992, 1926.
- N. Wagner, M. C. Mulder, M. S. Hsu, "A knowledge based control strategy for a biped," *Proceedings of the 1988 IEEE Conference on Robotics and Automation*, vol. 3, pp. 1520-4.
- H. Wang, T. T. Lee, W. A. Gruver, "A neuromorphic controller for a three-link biped robot," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 1, January/February 1992.
- C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. thesis, Cambridge University, Cambridge, England, 1989.
- B. Widrow, M. E. Hoff, "Adaptive switching circuits," *1960 IRE WESCON Convention Record*, New York: IRE, pp. 96-104
- R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3/4, pp. 229-256, May 1992.
- Y. F. Zheng and J. Shen, "Gait synthesis for the SD-2 biped robot to climb sloping surface," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 1, February 1990.
- Y. F. Zheng, "A neural gait synthesizer for autonomous biped robots," *IEEE International Workshop on Intelligent Robots and Systems, IROS*