ROBOTICS AND AUTONOMOUS SYSTEMS 22 (1997) 283-302

BIPED DYNAMIC WALKING USING REINFORCEMENT LEARNING

Hamid Benbrahim
GTE Laboratories Incorporated
40 Sylvan Road
Waltham, MA 02254
hbenbrahim@gte.com

Judy A. Franklin
Computer Science Department
Mount Holyoke College
South Hadley, MA 01075
jfranklin@mhc.mtholyoke.edu

Keywords: biped walking, reinforcement learning, robot learning, biped robot, legged robot.

ABSTRACT

This paper presents some results from a study of biped dynamic walking using reinforcement learning. During this study a hardware biped robot was built, a new reinforcement learning algorithm as well as a new learning architecture were developed. The biped learned dynamic walking without any previous knowledge about its dynamic model. The Self Scaling Reinforcement learning algorithm was developed in order to deal with the problem of reinforcement learning in continuous action domains. The learning architecture was developed in order 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 new modules that represent new knowledge, or new requirements for the desired task.

1 INTRODUCTION

The ultimate tangible goal of the research reported here is to achieve dynamic walking with a high level of adaptation and with minimal knowledge about the robot's dynamics. Learning and adaptation methods have been tried on real biped robots and showed promising results [22,36,37].

Although biped locomotion has been studied for a long time, it is only in the past twenty years, thanks to the development of fast 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 [16]. The control architecture had to make sure that the projection of the centre 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 [30]. The centre 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 [24], stair climbing [17,31], execution of successive flips [13], and even walking with no actuators [19].

Although there has been a large number of methods used to solve the problem of biped locomotion, it is difficult to detect a specific trend. We can however detect a major breakthrough that is definitely setting a new direction [5,22,27,32,36,37] and that is the introduction of learning and neural networks to biped locomotion. This has shown better results than conventional control methods.

In our research we have chosen to use reinforcement learning and CMAC neural networks.

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 [25,4] and optimal control, and has roots in the psychological study of classical conditioning [3,28].

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, [28] and Q learning, [33]. 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 an other. 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, [3,6,8]. 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, [7,11,12,35] 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.

In this paper we show how reinforcement learning is used within a modular control architecture to enable a biped robot to walk. First we describe the biped, then the learning architecture and finally the results. The Appendix presents some reinforcement learning

and neural networks fundamentals including the Self Scaling Reinforcement Algorithm (SSR) developed during this research.

2 THE BIPED

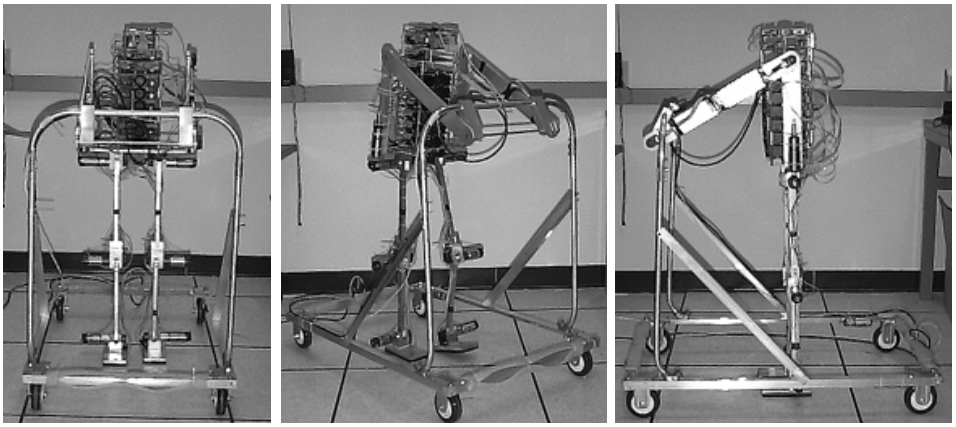

Figure 1: Biped views

The biped (Figure 1, Figure 2) is an aluminium 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. All joint axes of rotation are orthogonal to the sagittal plane.

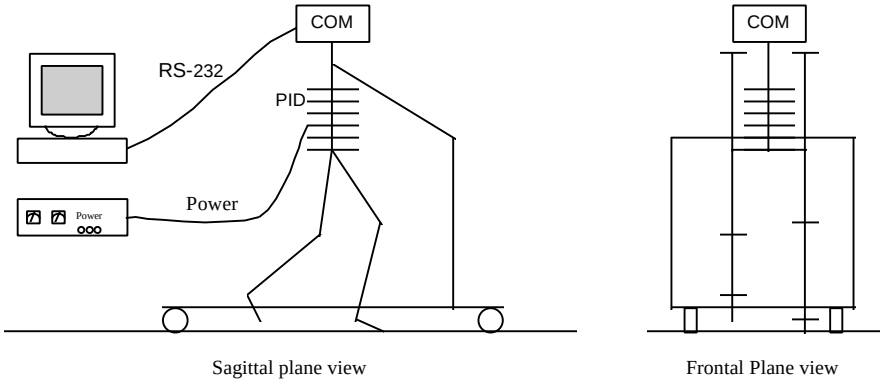

Figure 2: The Biped

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.

Joint positions are measured with optical encoders, with a resolution of 2000 counts per revolution. 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.

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 stabilise 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 aluminium 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, and 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, posture and foot positions can be easily determined from optical encoders that measure the hand and shoulder joint angles.

3 THE LEARNING ARCHITECTURE

This section describes the learning architecture that was developed to enable biped walking. The details of Actor-Critic reinforcement learning, Self-Scaling Reinforcement, Stochastic Real-Valued, and CMAC learning are presented in the appendix.

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.

A powerful learning architecture should be able to take advantage of any available knowledge [9]. 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 [8,14,23,26,34] 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 approach 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 centralises 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 3). 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 behaviour. 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.

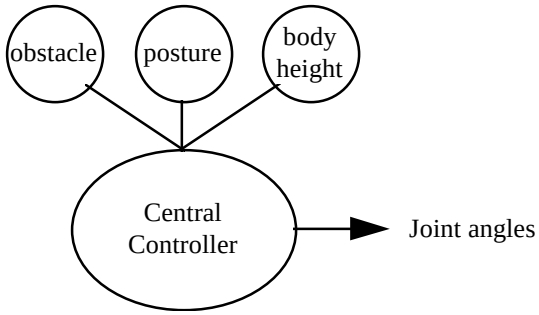

Figure 3: Architecture

The learning architecture used here is based on the assumption that there exists a nominal behaviour governed by a minimal number of inputs. The deviations from this behaviour are supposed to be small and caused by perturbations. Peripheral controllers individually act according to these perturbations.

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.

There are two ways in which information can be incorporated in this controller. 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.

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 behaviour, 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 behaviour 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 [10,15]. 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.

3.1 Central Controller

The central controller (Figure 4) 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 [36,37]. The small number of inputs allows the use of a small CMAC thus speeding up the learning and execution time for action generation.

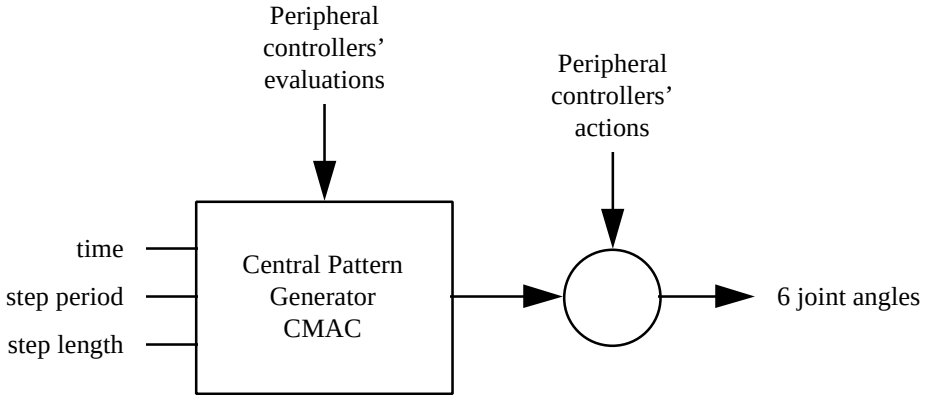

Figure 4: Central controller

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 3.3.

3.2 Peripheral controllers

There are four peripheral controllers: body posture, body height, step height, and energy expenditure.

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 peripheral controller (Figure 5) acts on both hip and ankle joints.

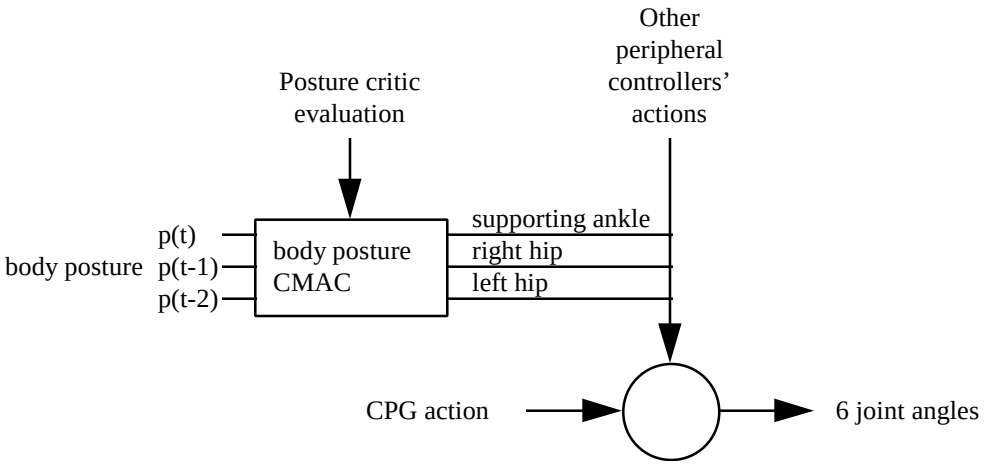

Figure 5: Body posture peripheral controller

Ankle joint torque can sufficiently control the posture only when the projection on the ground of the centre of gravity of the biped is on, or very close to, the centre 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.

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 behaviour. 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 10). Both CMACs use reinforcement learning with TD learning (see appendix).

Body height

The body height controller (Figure 6) determines the knee joint angle of the supporting leg.

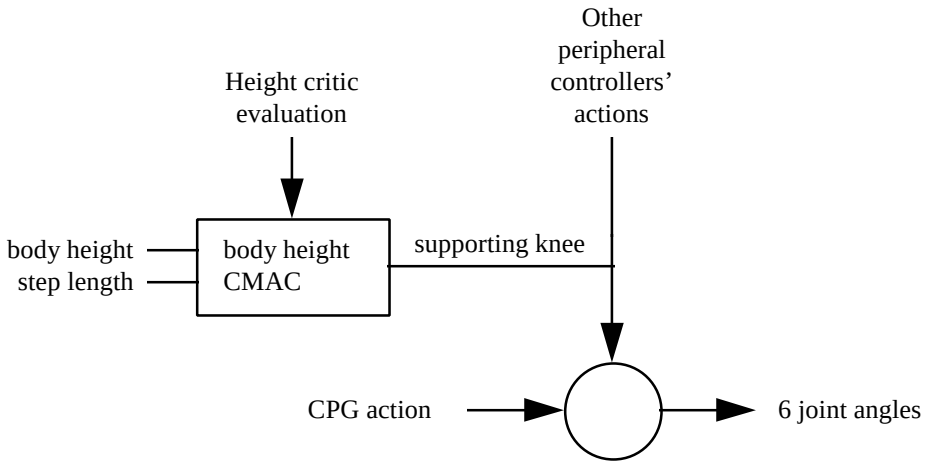

Figure 6: Body height peripheral controller

The CMAC's inputs are step length and body height. The weights are updated using reinforcement learning. 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.

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 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 7). 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.

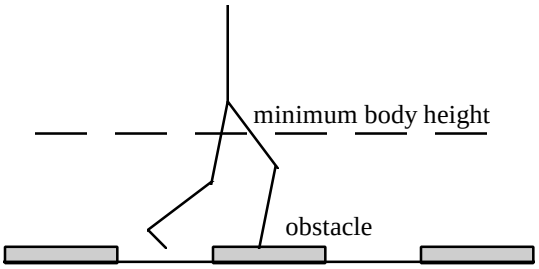

Figure 7: Step constraint

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.

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 minimise energy spending. This constraint guardian generates a reinforcement of the CPG's action based on the amount of energy spent.

3.3 Reinforcement learning

This section describes the learning process of the individual controllers described above.

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 reinforcement learning algorithm uses the actor-critic configuration [3]. It searches the action space using a Stochastic Real Valued (SRV) unit at the output [11]. 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), [7]. The reinforcement signal is generated using TD Learning [28,29] and the SSR algorithm.

The actor

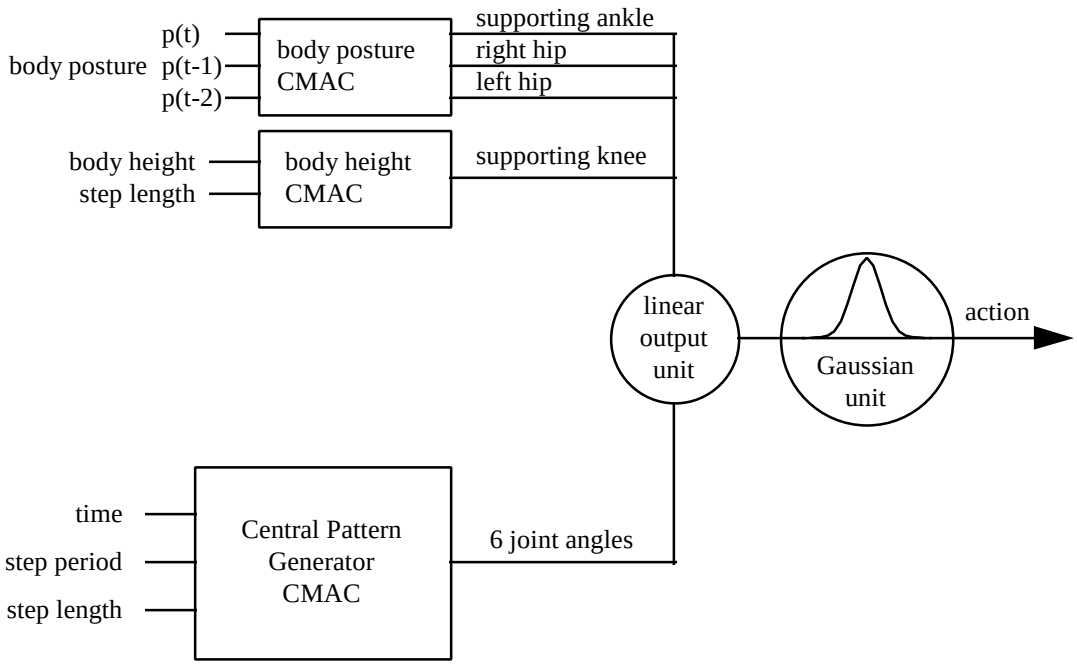

Figure 8: The actor

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 8 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.

The CMAC weights of all of the controllers including the CPG are updated using the same equations (equations:(1,(2). Figure 9 shows the actor configuration of Figure 8 with focus on only one CMAC.

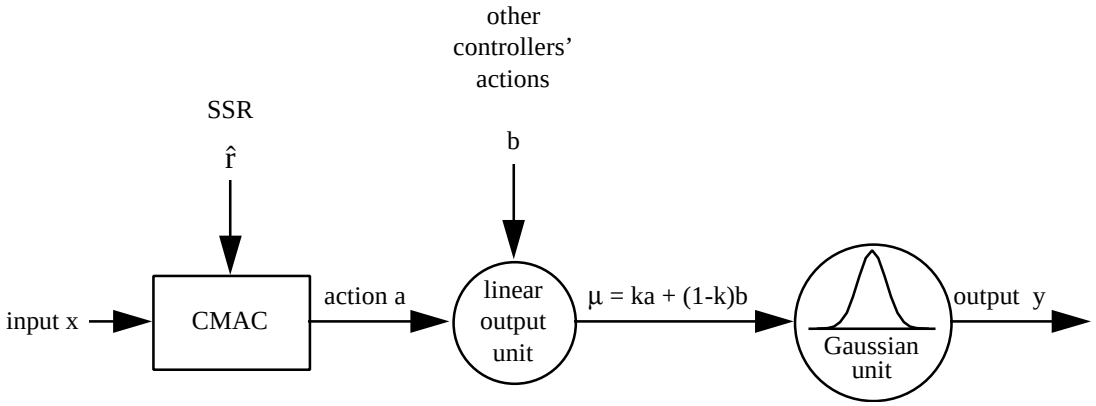

Figure 9: General actor configuration

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

$$w(t+1) = w(t) + \alpha \hat{r}(y-a)\frac{\partial a}{\partial w} \qquad (1)$$

$$\sigma(t+1) = \gamma\sigma + (1-\gamma)(rmax - rmin) \qquad (2)$$

Where the reinforcement $\hat{r}$ is provided by the SSR equations (see appendix).

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 behaviour 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 9 are zero. Equation (1) becomes then

$$w(t+1) = w(t) + \alpha \hat{r}(b-a)\frac{\partial a}{\partial w} \qquad (3)$$

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.

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.

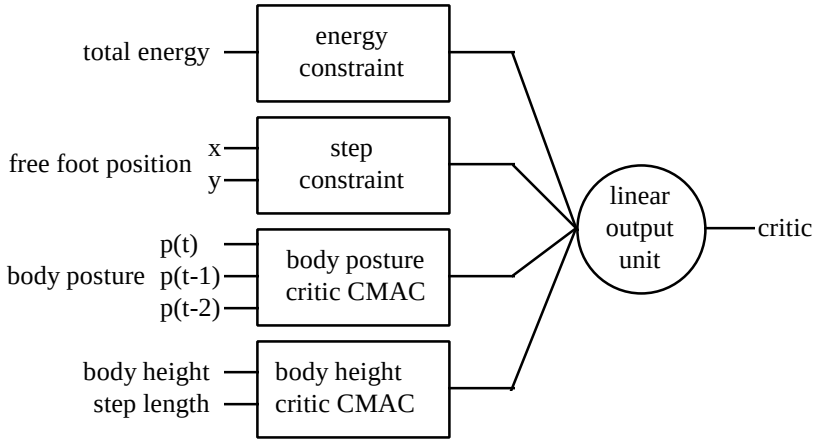

Figure 10: The CPG critic

Figure 10 shows the CPG critic configuration. The posture and height critics use CMAC neural networks. These networks are trained using TD learning as described in the appendix. Each critic generates a reinforcement signal as follows:

$$\hat{r} = r + \gamma p(x_k, t) - p(x_{k-1}, t-1) \qquad (4)$$

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

Energy and foot step critics are predefined. The energy critic generates a reinforcement value that is proportional to the negative of the energy and the foot step critic generates a -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%.

4 EXPERIMENTS & RESULTS

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 [18] 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 behaviour 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. A low pass filter is placed at the robot joint level to minimise jerkiness, and only relative actions are generated and accumulated over time instead of absolute actions, assuming that the nominal action is a continuous function.

4.1 Pre-training

CPG CMAC pre-training

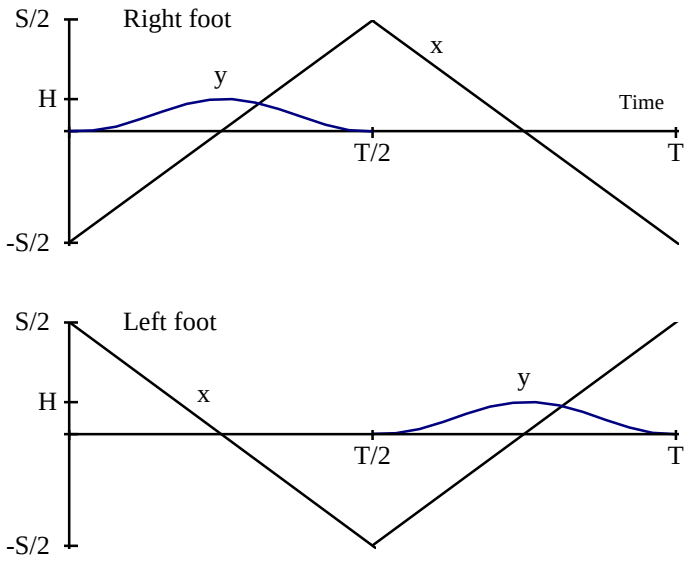

Figure 11: Pre-training foot positions

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 feet positions are always in quadrature.

Figure 11 shows desired foot positions where T is the walking period, S is the step length and H is the step height.

The constant body height is chosen so that the leg is completely extended at the end of each step. 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 generalisation 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.

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.

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 behaviour. The role of this teacher is to bring the CMAC's weights to the neighbourhood of an optimal solution.

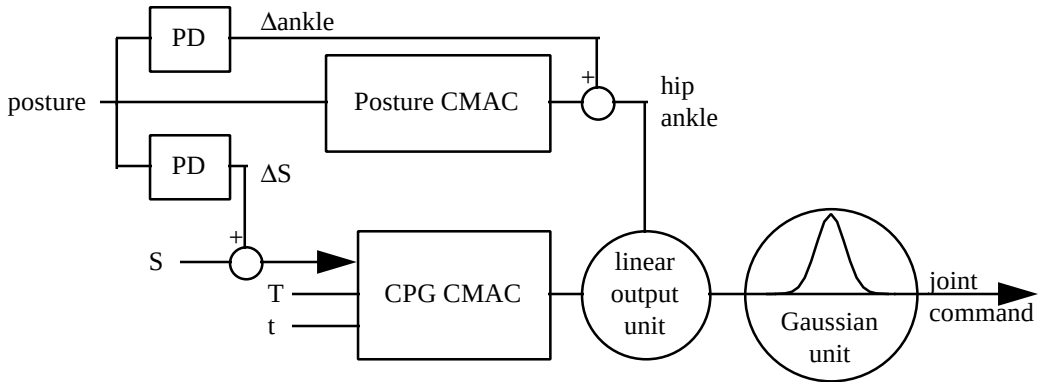

Figure 12: Posture CMAC pre-training

Figure 12 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 generalisation 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 +/-π/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.

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.

4.2 The Simulator

The simulator is mainly used to pre-train all of the CMAC neural networks in order to speed up learning. 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.

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.

A failure signal is generated every time the posture exceeds +/- $\pi/5$, 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 +/- $\pi/20$, and the height CMAC is activated when the difference between the body height and the target height exceeds +/- 2cm. The target height is determined by computing the running average of the height. An optimum height is reached once the CPG has learned.

4.3 Results

Simulator learning curve

The learning curve in Figure 13 shows the average successful walking time of the simulated biped. 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.

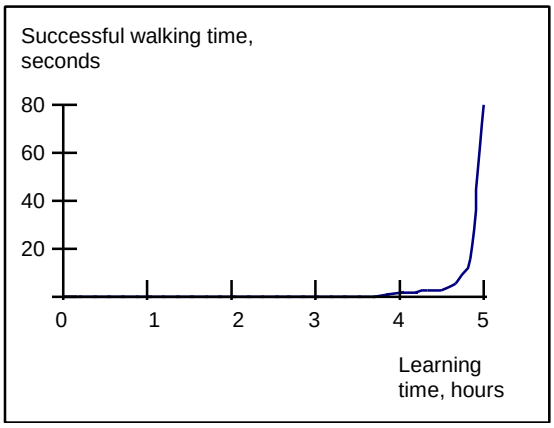

Figure 13: CPG learning curve

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.

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. 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. The bang-bang action is activated only when the posture deviation exceeds +/-π/10.

Figure 14 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.

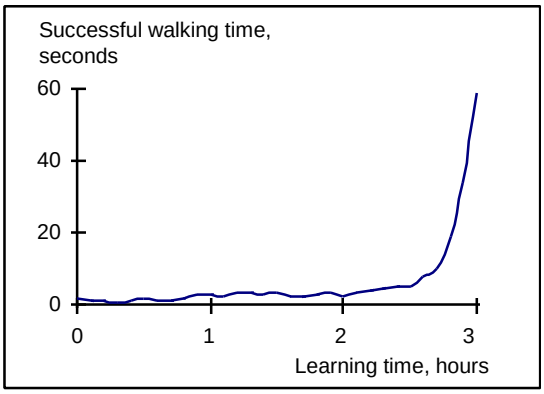

Figure 14: Extra learning curve

Hardware 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. The biped was not able to walk reliably at higher walking speed or longer steps.

Figure 15 and Figure 16 show the right and left foot positions. These are similar to the pre-training foot positions shown in Figure 11. Notice that the x foot trajectory averages are less than zero. This means that the robot walks with its feet slightly lagging behind. This behaviour 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 behaviour is obvious because moving the supporting leg forward would cause instability. One plausible explanation of the first behaviour 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.

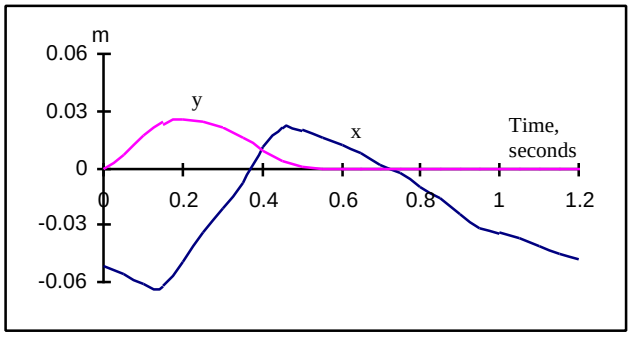

Figure 15: Real foot positions: x_r, y_r

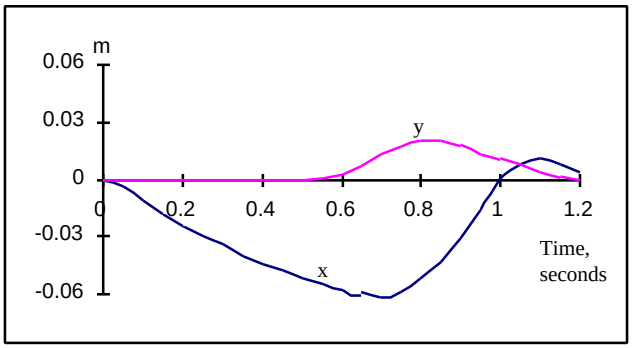

Figure 16: Real foot positions: x_l, y_l

Figure 17 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 behaviour 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.

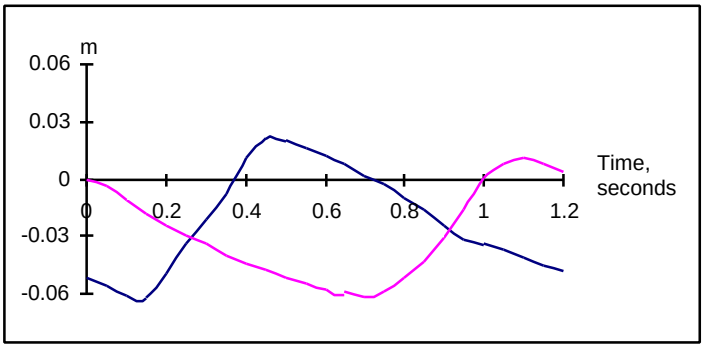

Figure 17: Real foot positions: x_r, x_l

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 18 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 aluminium 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.

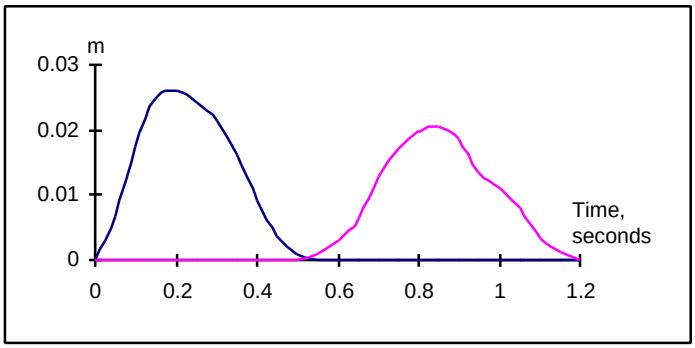

Figure 18: Real foot positions: y_r, y_l

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 behaviour, thus smooth trajectories.

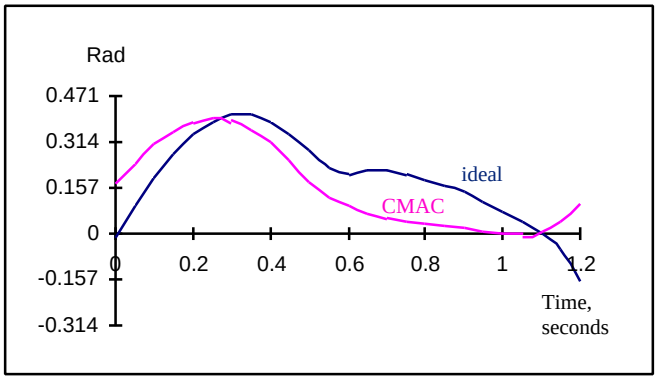

Figure 19: Ideal vs CPG joint positions: hip

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

Figure 20 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.

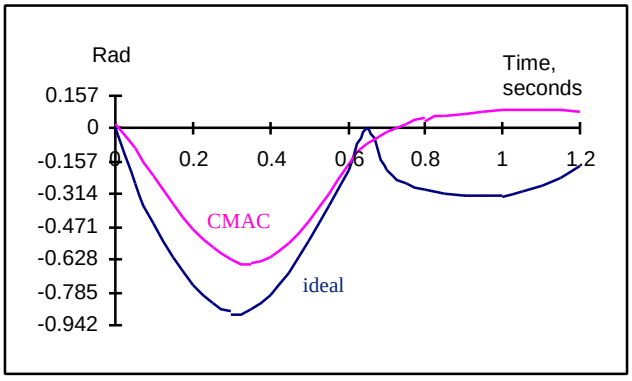

Figure 20: Ideal vs CPG joint positions: knees

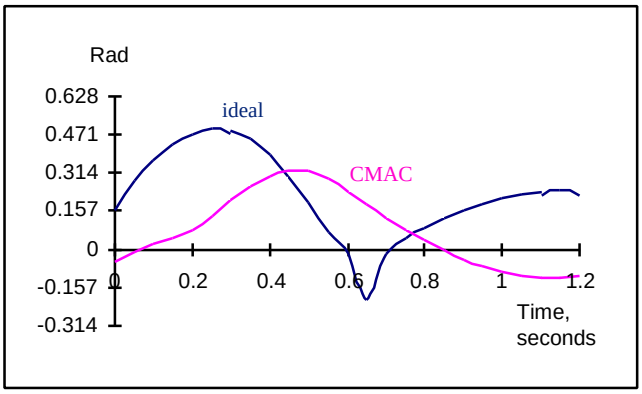

Figure 21: Ideal vs CPG joint positions: ankles

Figure 21 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.

Body height

Figure 22 show the biped's body height over an 18 second walk. The body height is not constant because the supporting knee is constantly locked. 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.

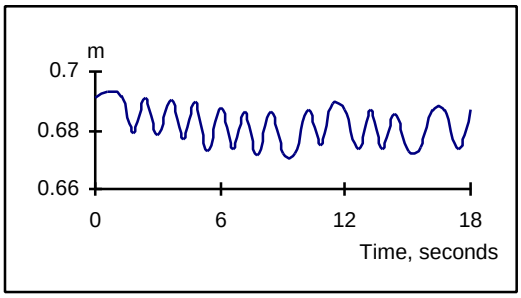

Figure 22: Body height

Body posture

Figure 23 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 -π/20 line indicates the threshold when the posture peripheral controller intervenes.

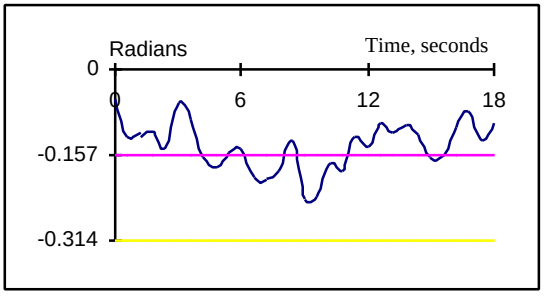

Figure 23: Body posture

Posture intervention

Figure 24 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.

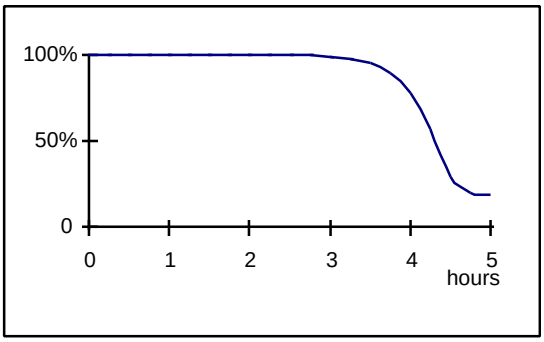

Figure 24: Posture constraint intervention frequency

5 CONCLUSION

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.

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.

6 REFERENCES

[1] 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.

[2] J. S. Albus, "Data storage in the cerebellar model articulation controller," *ASME Journal of Dynamic systems, Measurements, and Control*, pp. 228-233, September 1975.

[3] 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.[4] A. G. Barto, S. P. Singh, "Reinforcement learning and dynamic programming," *Proceedings of the 6th Yale Workshop on Adaptive and Learning Systems*, 1990.

[5] J. S. Bay and H. Hemami, "Modeling of a neural pattern generator with coupled non-linear oscillators," *IEEE Transactions on Biomedical Engineering*, vol. BME-34, no. 4, April 1987.

[6] 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.

[7] 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.

[8] J. A. Franklin, "Refinement of robot motor skills through reinforcement learning," *Proceedings of the 27th IEEE Conference on Decision and Control*, December 1988.

[9] 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.

[10] E. C. Freuder, R. J. Wallace, "Partial constraint satisfaction," *Artificial Intelligence*, vol. 58, pp. 21-70, 1992.

[11] V. Gullapalli, "A stochastic reinforcement learning algorithm for learning real-valued functions," *Neural Networks*, vol. 3, pp. 671-692, 1990.

[12] 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.

[13] J. K. Hodgins, M. H. Raibert, "Biped Gymnastics," *The International Journal of Robotics Research*, vol. 9, no. 2, April 1990.

[14] 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.

[15] M. I. Jordan, "Constrained supervised learning," *Journal of Mathematical Psychology*, vol. 36, pp. 396-425, 1992.

[16] I. Kato, S. Ohteru, 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.

[17] 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.

[18] 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.

[19] T. McGeer, "Passive dynamic walking," *The International Journal of Robotics Research*, vol. 9, no. 2, April 1990.

[20] D. Michie, R. Chambers, "Boxes: an experiment in adaptive control," *Machine Intelligence*, E. Dale, D. Michie, editors. Edinburgh, UK: Oliver and Boyd, 1968.

[21] 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.

[22] 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.

[23] K. S. Narendra, J. Balakrishnan "Intelligent control using switching and tuning," *Proceedings of the 8th Yale Workshop on Adaptive and Learning Systems*, 1994.

[24] 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.

[25] 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.

[26] J. Sklansky, "Learning systems for automatic control," *IEEE Transactions on Automatic Control*, vol. AC-11, pp. 6-19, 1966.

[27] S. Stitt, Y. F. Zheng, "Distal learning applied to biped robots,". Y. F. Zheng editor. *Recent Trends in Mobile Robots*, World Scientific, 1993.

[28] R. S. Sutton, "Temporal Credit Assignment in Reinforcement Learning," Ph.D. thesis, University of Massachusetts, Amherst, MA, 1984.

[29] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9-44, 1988.

[30] 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.

[31] 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.

[32] 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.

[33] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. thesis, Cambridge University, Cambridge, England, 1989.

[34] B. Widrow, M. E. Hoff, "Adaptive switching circuits," *1960 IRE WESCON Convention Record*, New York: IRE, pp. 96-104

[35] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3/4, pp. 229-256, May 1992.

[36] 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.

[37] Y. F. Zheng, "A neural gait synthesizer for autonomous biped robots," *IEEE International Workshop on Intelligent Robots and Systems*, IROS '90.

7 APPENDIX: THE LEARNING ALGORITHMS

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 behaviour, 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. We use TD learning described below.

Temporal Differences (TD) learning [3,28] is used to solve the problem of delayed rewards in reinforcement learning. It can also be used in a variety of prediction tasks [22]. 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:

$$\hat{r} = r + \gamma p(x_k, t) - p(x_{k-1}, t-1) \qquad (5)$$

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:

$$p(x_k, t) = p(x_k, t_l) + \beta\hat{r} \qquad (6)$$

where β is a positive decay factor < 1 and t_l is the time when state x_k was last visited.

The actor uses a modified The Stochastic Real-valued (SRV) Algorithm [11]. For every input vector x, the SRV algorithm generates $\mu(x)$ and $\sigma(x)$. A Gaussian random number generator with mean $\mu(x)$ and standard deviation $\sigma(x)$ generates the action y(x).

$\mu(x)=wx$. The weight vector w, and $\sigma(x)$ are updated by:

$$w_\mu(t+1) = w_\mu(t) + \alpha(r - \bar{r})\frac{y-\mu}{\sigma}x \qquad (7)$$

$$\sigma = k(\max(r) - \bar{r}) \qquad (8)$$

where k is a multiplying factor, r is the reinforcement and $\bar{r}$ is the average of r. When the system has learned, the average reinforcement becomes close to the maximum reinforcement and σ becomes very small. This means that the action will be essentially equal to μ, and thus the search will be stopped.

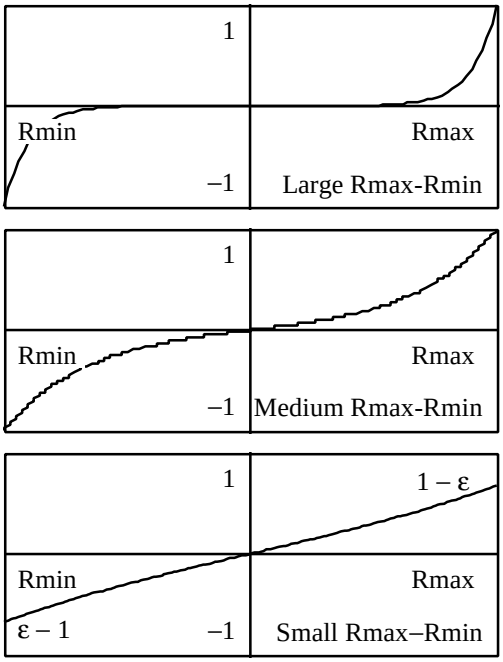

Figure 25: Self Scaling Reinforcement

The Self Scaling Reinforcement algorithm (SSR) [7] is a new reinforcement learning algorithm that 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, rmax and rmin respectively, and computes a scaled reinforcement as follows:

$$\hat{r} = \exp(r - rmax) - \exp(rmin - r) \qquad (9)$$

Notice that if r = rmax, then, $\hat{r} = 1 - \exp(rmin - rmax) = 1 - \varepsilon$ and if r = rmin, then $\hat{r} = \exp(rmin - rmax) - 1 = \varepsilon - 1$, ε is very small when rmin-rmax << 0; consequently, it can be ignored for clarity purposes.

Figure 25 shows three graphs of the SSR as a function of r for large, medium and small rmax-rmin, where ε is not ignored. Notice that ε increases as rmax-rmin decreases. For large values of rmax-rmin, only extreme reinforcement is taken into account. This means that the system uses very low resolution for the reinforcement signal in the beginning. As rmax-rmin diminishes the SSR function becomes more linear, thus a finer resolution is used. As rmax-rmin converges toward zero, the SSR function converges toward the x axis. The learning then stops. If a new reinforcement occurs outside the [rmin, rmax] segment, then rmax-rmin automatically increases and more learning ensues.

rmax and rmin are computed according to 4 equations:

$$rmax(t+1) = \max\{rmax(t), r\} \qquad (10)$$

$$rmin(t+1) = \min\{rmin(t), r\} \qquad (11)$$

$$rmax(t+1) = l.rmax(t+1) + (1 - l)r \qquad (12)$$

$$rmin(t+1) = l.rmin(t+1) + (1 - l)r \qquad (13)$$

where λ is a positive number < 1. The SSR Equations work together as follows. rmax increases as the system gets higher reinforcement values, rmin decreases as the system gets lower reinforcement values. The difference between rmax and rmin converges, as the system learns, toward zero. rmin increases when low reinforcement values are infrequent and rmax is deliberately decreased in order to filter out large spurious reinforcement values.

Figure 26 shows graphically the expected behaviour of rmax and rmin as the system learns.

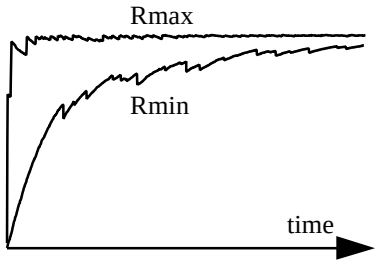

Figure 26: SSR convergence

The algorithm's adaptation equations are

$$w_\mu(t+1) = w_\mu(t) + \alpha\hat{r}(y-\mu)\frac{\partial\mu}{\partial w_\mu} \qquad (14)$$

$$\sigma(t+1) = \gamma\sigma + (1-\gamma)(rmax - rmin) \qquad (15)$$

where γ is a positive number < 1. It is used as an averaging factor.

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

7.1 CMAC Neural Networks

The Cerebellar Model Arithmetic Computer (CMAC) [1,2] is an associative neural network that tries to mimic the biological sensory neurones 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 generalise what it has learned about a certain region to its neighbourhood.

A CMAC can be configured in several ways [21]. The simplest form of CMAC is achieved by using several "boxes" configuration [20] 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 generalisation of the learning from one region to another. The individual networks can be called layers for simplicity.

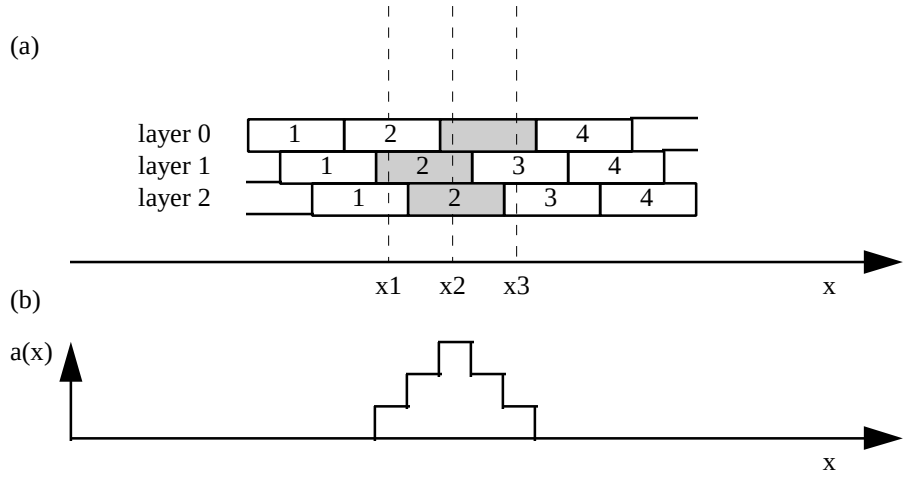

Figure 27: CMAC neural network input/output representation

The example of Figure 27(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 x1, x2, and x3 are as follows:

$$a(x1) = \frac{a_{0,2} + a_{1,2} + a_{2,1}}{3} \qquad (16)$$

$$a(x2) = \frac{a_{0,3} + a_{1,2} + a_{2,2}}{3}$$

$$a(x3) = \frac{a_{0,3} + a_{1,3} + a_{2,3}}{3}$$

Suppose that we start with all of the weights set to zero. Then we present the CMAC with the input x2 exclusively and make it learn the optimum action a(x2). We then present the CMAC with the inputs x1 and x3 without performing any learning. The actions related to x1 and x3 will not be zero because of the boxes that they share with the input x2. Figure 27(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 behaviour from one region is generalised to neighbouring 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 centre 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.