CMAC Algorithm for Motion Control in the Prensence of Friction

R.H.A. Hensen

June 21st, 1996

WFW Stage Report no.96.099

Professor: Prof.dr.ir. J.J. Kok
Tutor: Dr.ir. M.J.G. van de Molengraft

Eindhoven University of Technology
Department of Mechanical Engineering

Section: Fundamentals
Group: System and Control Engineering

CMAC Algorithm for Motion Control in the Presence of Friction

R.H.A. Hensen

June 21st, 1996

Abstract

The objective of control algorithms is to minimize the difference between the desired and performed motion. Mathemattically, the dynamic system, which has to be controlled, can be nonlinear. Difficulties in designing control algorithms are introduced by these nonlinearities. Though human beings handle easily these problems without extensive computations. The Cerebellar Model Articulation Controller (CMAC) algorithm is a mathematical model of the human cerebellar, which controls the movements of human beings.

In this paper the CMAC algorithm is tested on the inverted pendulum. The friction in the actuator introduces the nonlinearity of the dynamic system. The CMAC algorithm can be added (i) on-line or (ii) off-line parallel to a conventional controller.

In the on-line configuration the conventional controller uses only errors, such as position and velocity errors, to compute the control action (e.g. PID controller), The output of the conventional controller is the control objective which has to be minimized by the CMAC algorithm.
On the inverted pendulum this configuration is tested with a PID controller as the conventional controller. The results of the on-line configuration are compared with the results performed with a PID controlller. The results of the CMAC algorithm are more accurate than the PID controller. This means that the CMAC algorithm can indeed handle friction. Possibly other nonlinearities can be faced well with the CMAC algorithm.

In the off-line configuration the CMAC algorithm is able to copy each conventional controller for a certain motion. The performance of the CMAC algorithm is maximal the same as the conventional controller. In other terms the CMAC algorithm fits the data of the conventional controller for that motion.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The problem of controlling a manipulator is that of finding the right control signal for each joint actuator at every point in time and under every set of conditions. In order that the manipulator carries out any movement, it is necessary to drive each joint through a trajectory (a sequence of positions) as a function of time. The control signal is also a function of other variables such as:

- Position, velocity and acceleration in most, or all, of the joints.
- Force and touch signals (skin in human control systems) from various points on the manipulator.
- Visual or other feedback measurements concerning the position of the end point (end-effector).
- Data of bending, twisting and friction in various components of the manipulator.
- The drive signals depend also on higher level input variables which identify the desired trajectory of the end point or the movement of the joints.

Though the particular task which has to be performed may be stated in simple terms, the solution is often very complicated.

For controlling mechanical manipulator systems, the trajectory each joint has to make to produce a certain motion of the end-effector is solved by computations based on relationships between physical elements of the manipulator itself. These relationships are based on more or less idealized mathematical formulations. Such formulations usually take into account only the joint velocities and angles. It is possible to include gravity loading and inertial forces. However, in the real world the variables and nonlinearities introduce a computational problem which cannot be solved appropriately by these mathematical formulations.

When one examines nature and observes the movements made by biological organisms, one will belief that the solution of mathematical eqautions is not the adequate method for producing drive signals for a manipulator control problem.

In human control systems there are two levels of data processing, the conscious level and the subconscious level. At the conscious level there are two relevant input variables to the manipulator control computation. First, the task that has to be accomplished, such as "pick up a glass" in the range of tasks as "brush teeth", "comb hair" or "slap". Second, it is necessary to measure the position of the hand (end point) relative to the glass (desired trajectory) and compute the direction vector which is required to move the hand towards the glass. The detailed computation of the action of every muscle for this movement is done at the subconscious level.

One part of the brain that seems to be intimately involved in motor control processes is the cerebellum [1]. Input to the cerebellum arrives in the form of sensory and feedback from the muscles, joints and skin together with the higher level commands concerning the desired trajectory. According to the theory, this input constitutes an address, which contains the appropriate muscle actuator signals required to carry out the desired movement. At each point in time the input addresses an output which drives the muscle control circuits. The resulting motion produces a new input and the process is repeated. The result is the desired trajectory of the manipulator through space. At each point on the trajectory the state of the manipulator is sent to the cerebellum as input, and the cerebellar memory responds with the actuator signals which drive the manipulator to the next point on the trajectory.

This paper describes the control problem of a practical manipulator, the inverted pendulum. The inverted pendulum is controlled with a cerebellum controller (Cerebellar Model Articulation Controller). The inverted pendulum is a system with two degrees of freedom and only one actuator. The two degrees of freedom are the two angles which are measured on-line. Besides the pendulum itself, the static (Coulomb) and kinetic friction in the actuator makes the dynamic system nonlinear. Conventional controllers, using these dynamic models, cannot solve the control problems appropriately. The purpose of this work is to see whether the CMAC algorithm can work on this inverted pendulum in the presence of friction and second compare the results with already existing friction compensating algorithms.

Chapter 2

Theory

The CMAC algorithm will be explained in this chapter. In particular the algorithm applied for the experiment with the inverted pendulum will be discussed.

2.1 The CMAC System

Every control algorithm has input and output vectors. The input vector $\mathcal{S}$ is a R-ary vector. An input vector $\mathcal{S}$ in the cerebellum produces an association cell vector $\mathcal{A}$ which is a binary vector. This association cell vector addresses the weight matrix W and produces a response vector $\mathcal{P}$.

Mathematically, the cerebellum may be represented by a pair of mappings

$$\begin{aligned} f &: \mathcal{S} \rightarrow \mathcal{A} \\ g &: \mathcal{A} \rightarrow \mathcal{P} \end{aligned} \qquad (2.1)$$

where

$$\begin{aligned} \mathcal{S} &= \text{input vector} \\ \mathcal{A} &= \text{association cell vector} \\ \mathcal{P} &= \text{output vector} \end{aligned}$$

The function f is generally fixed, but the function g depends on the value of the weights which may be modified during the data training process.

In the practical system of the inverted pendulum the input vector $\mathcal{S}$ and the output vector $\mathcal{P}$ can be defined as followed.

- The R -ary input vector $\mathcal{S}$ could contain, for instance, the
  1. position of the pendulum (β)
  2. velocity of the pendulum ($\dot{\beta}$)
  3. velocity of the motor ($\dot{\alpha}$)
  4. position error of the pendulum ($\epsilon_\beta = \beta_{desired} - \beta$)
  5. velocity error of the pendulum ($\dot{\epsilon}_{\dot{\beta}} = \dot{\beta}_{desired} - \dot{\beta}$)
  6. velocity error of the motor ($\dot{\epsilon}_{\dot{\alpha}} = \dot{\alpha}_{desired} - \dot{\alpha}$)
- The response output vector $\mathcal{P}$ contains the applied momentum of the actuator U [Nm].

When an input vector $\mathcal{S}$ is presented to the sensory cells of the cerebellum, it is mapped into an association cell vector $\mathcal{A}$. The non-zero elements of $\mathcal{A}$ are defined as A^* as shown in Figure 2.1. The response cell of the cerebellum

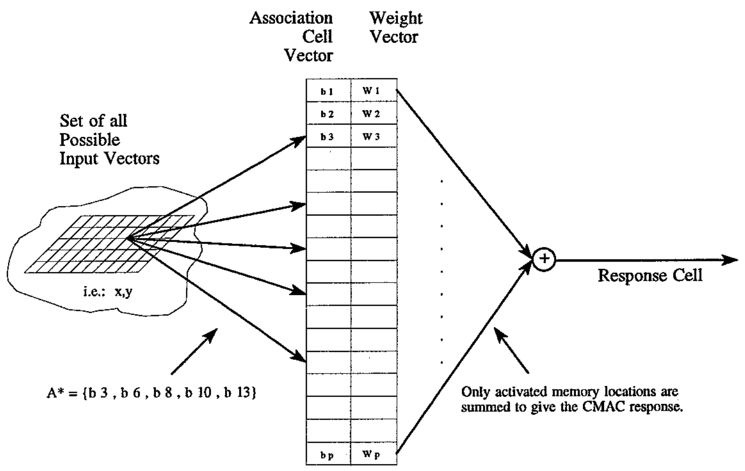
Association Cell Vector
Weight Vector
Set of all Possible Input Vectors
i.e.: x,y
A* = {b 3 , b 6 , b 8 , b 10 , b 13}
b1
b2
b3
bp
W1
W2
W3
Wp
Response Cell
Only activated memory locations are summed to give the CMAC response.

Figure 2.1: The CMAC algorithm

sums the values of the weights addressed by the active association cells. Only the nonzero elements A^* affect this sum, which is the output vector $\mathcal{P}$. If an output vector $\mathcal{P}$ is desired to be changed, under a given input vector $\mathcal{S}$, then one **only** adjusts the weights of the association cells in A^*.

The cerebellum does not have sufficient association cells to address a unique cell or a group of cells for any possible input vector $\mathcal{S}$. This means that an association cell may be addressed by different input vectors $\mathcal{S}$. For example, if there are two input vectors $\mathcal{S}_1$, $\mathcal{S}_2$ and the sets nonzero association cells are A_1^* and A_2^*, it is possible that there are common association cells as shown in Figure 2.2. This is called overlap or cross-talk between input

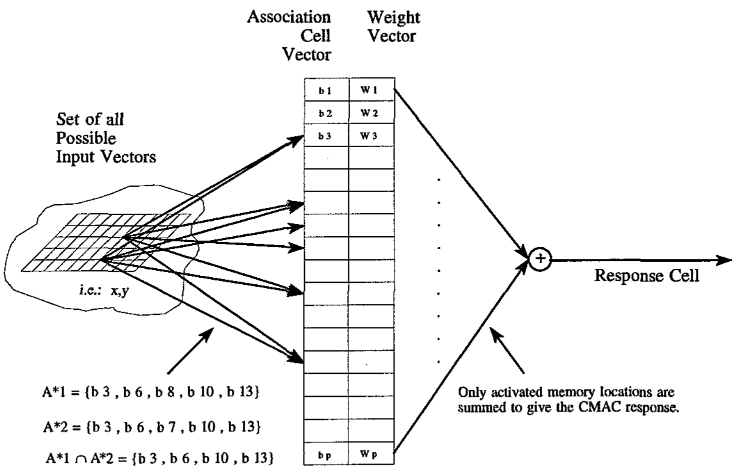
Association Cell Vector
Weight Vector
Set of all Possible Input Vectors
i.e.: x,y
b1
b2
b3
bp
w1
w2
w3
wp
+
Response Cell
Only activated memory locations are summed to give the CMAC response.
A*1 = {b 3 , b 6 , b 8 , b 10 , b 13}
A*2 = {b 3 , b 6 , b 7 , b 10 , b 13}
A*1 ∩ A*2 = {b 3 , b 6 , b 10 , b 13}

Figure 2.2: The overlap in two input vectors

vectors which in some cases is beneficial and in other cases leads to serious problems.

If it is desired that the output response cell produces for $\mathcal{S}_1$ the same output as for $\mathcal{S}_2$ the overlap is beneficial. In that case, the weights addressed by input vector $\mathcal{S}_2$ should not be adjusted to produce the same output vector. This property is called generalization and means that the cerebellum generalizes from one learning experience to another. The generalization size is the number of non-zero elements of $\mathcal{A}$ or the number of elements in A^*.

On the other hand, if it is desired that the output response cell produces for $\mathcal{S}_2$ a different output then for $\mathcal{S}_1$, the overlap $A_1^* \wedge A_2^*$ introduces difficulty. The adjustments of the weights (of the set nonzero association cells) in A_2^* will upset the weights in A_1^*. The output vector of $\mathcal{S}_1$ will be changed after adjusting the weights for the output vector $\mathcal{S}_2$. This is called learning interference, which can be overcome by repeated iteration of the data storage algorithm for $\mathcal{S}_1$ and $\mathcal{S}_2$. This iteration leads to large weights outside the overlap $A_1^* \wedge A_2^*$, but *not* in the overlap so that the different output vectors can be obtained for $\mathcal{S}_2$ as well as for $\mathcal{S}_1$. The ability to produce dissimilar outputs for different input vectors, is derived from the difference between A_1^* and A_2^*. In general, the smaller the overlap $A_1^* \wedge A_2^*$, the easier it is to find a set of weights which will produce a dissimilar output for $\mathcal{S}_2$ as well as for $\mathcal{S}_1$.

The control function for a manipulator is typically a rather smooth continuous function. This means that for every point in input-space (state of the input vector $\mathcal{S}$) which requires a certain response, there is a small neighbourhood of input-space points around that point for which nearly the same response is required. Within that neighbourhood, the control system should produce approximately the same response. On the other hand, the control function should be independent for widely seperated points in input-space. In more precise terms, if two input vectors $\mathcal{S}_i$ and $\mathcal{S}_j$ have a small input-space distance, then the overlap $A_i^* \wedge A_j^*$ should be large. On the contrary, if $\mathcal{S}_i$ and $\mathcal{S}_j$ have a large input-space distance, $A_i^* \wedge A_j^*$ should be small. The Hamming input-space distance is defined as

$$H_{ij} = \sum_{k=1}^{N} \mid s_{ik} - s_{jk} \mid \tag{2.2}$$

where s_{ik} are the components of the input vector $\mathcal{S}_i$ and N is the dimension of $\mathcal{S}_i$:

$$\mathcal{S}_i = (s_{i1}, s_{i2}, \ldots, s_{iN})$$

$$\mathcal{S}_j = (s_{j1}, s_{j2}, \ldots, s_{jN})$$

If H_{ij} is small, $A_i^* \wedge A_j^*$ should be large. As H_{ij} grows larger, $A_i^* \wedge A_j^*$ should get smaller until, at some value of H_{ij}, $A_i^* \wedge A_j^*$ should be zero.

In the cerebellum it is believed that for any input vector $\mathcal{S}$, $\mid A^* \mid$[1] is less than one percent of the total number of association cells [1]. $\mid A \mid$ is the number of association cells physically implemented by the CMAC algorithm. $\mid A \mid$ is typically chosen as at least 100 times $\mid A^* \mid$.

There are many ways to address the association vector with a given input vector. A conventional way of addressing the association vector would be to assign one association cell for each input space. For example, if we have an input vector of dimension $N = 7$, with resolution of each dimension $R_i = 1000$, the association vector would require 10^{21} cells.

With the CMAC algorithm, as described above, many cells are addressed for each input vector. The number of cells assigned for an input vector is the generalization size $\mid A^* \mid$. Each state is mapped into $\mid A^* \mid$ cells, and the next closest input vector would have $\mid A^* - 1 \mid$ cells in common. Therefore the number of cells needed is reduced to $\mid A^* \mid (R_i / \mid A^* \mid)^N$. If for the previous example each input vector was mapped into $\mid A^* \mid = 100$ cells, only $100 * (10)^7$ or 10^9 association cells would be required. However, these numbers of cells are still very large for implementation in an experimental environment.

[1] $\mid X \mid$ is defined as the number of elements in X.

2.2 The CMAC Mapping Algortihm

The CMAC mapping algorithm [2], corresponding to the theory above, is defined as a series of mappings from an input vector $\mathcal{S}$, to an output vector $\mathcal{P}$,

$$\mathcal{P} = H(\mathcal{S}) \tag{2.3}$$

where $H(\mathcal{S})$ represents the overall mapping. Each dimensional mapping $H(\mathcal{S})$ consists of three smaller mappings in the following order,

$$\mathcal{S} \rightarrow \mathcal{M} \rightarrow \mathcal{A} \rightarrow \mathcal{P}$$

where $\mathcal{S}$ is an input vector, containing both desired and feedback information, $\mathcal{M}$ is the set of quantization functions used to encode $\mathcal{S}$, $\mathcal{A}$ is the set of memory cells addressed by $\mathcal{M}$, and $\mathcal{P}$ is the output vector.
As mentioned above, there is still a memory problem. To face this problem a routine called hash coding [3] can be used. The hash coding routine uses the large memory set $\mathcal{A}$ as input and maps this in a smaller physical memory. This may introduce collision of the data, but with the right setting of the parameters this can be optimized.

For the inverted pendulum this algorithm is not used, but another solution is tested [4]. In this paper, each component X_i [2] of the input vector is mapped into a one-dimensional memory table. The input vector dimension is N. The total available physical memory size is $\mid A \mid$ and the generalization size is $\mid A^* \mid$. If the available memory is shared equally by the dimension of the input vector, each component of the input vector would have $\mid A_i \mid = \mid A \mid /N$ memory cells available. The generalization size for a sampled component X_i of the input vector would be $\mid A_i^* \mid = \mid A^* \mid /N$. Each component of the input vector X_i is quantized to a value X_{iq} with a resolution of 1 part in $\mid A_i \mid$. The memory cells addressed for each input vector component X_i are, the memory cell addressed by X_{iq}, plus the next $\mid A_i^* \mid -1$ cells, as shown in Figure 2.3.

This CMAC mapping algorithm also has the advantage of further reductions in memory size. The number of memory cells needed with this algorithm is only $R_i * N$. For the previous example, there are only 7000 memory cells required instead of 10^9 (other CMAC-algorithm) or 10^{21} cells. Notice that the memory reduction increases especially for large input dimensions N. On the other hand, it is not said that this CMAC algorithm achieves the same accuracy as the other algorithms. This CMAC approach may need a larger sampling resolution R_i to achieve the same accuracy. For a specific

[2]i is an input vector component $i = 1, \ldots, N$

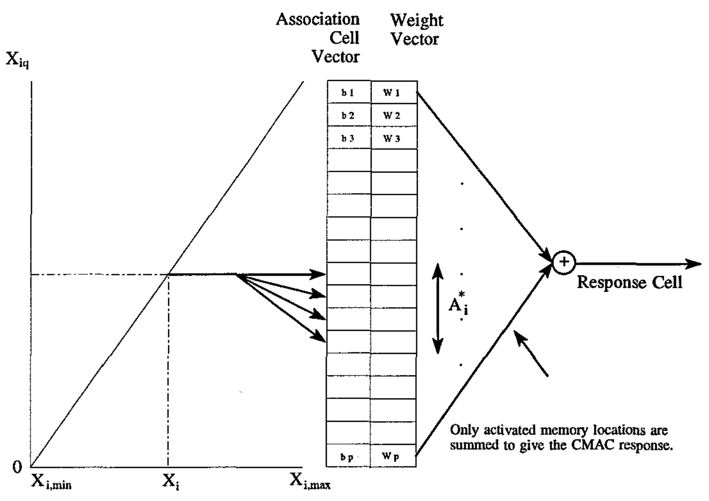

Figure 2.3: Linear Quantization of the input component X_i

application this comparison may not indicate the actual difference in memory requirements.

The essential part of this mapping algorithm is the quantization of the input vector. The quantization of the input vector must be such that it best utilizes the available memory cells. Both, for each component of the input vector and every specific application, different quantization functions $f(X_i)$ may be needed. Two quantization functions can be distinguished , i.e. (i) the linear quantization and (ii) the nonlinear quantization. The quantization functions used for this application are given later.

2.3 Shaping Input-Space Neighbourhoods

The components, $X_{i,1}$ and $X_{i,2}$, of two input vectors S_1 and S_2 are in the same neighbourhood if $A_{i,1}^* \wedge A_{i,2}^*$ is not null, thus there is overlap [2]. The size of the neighbourhood for each component of an input vector depends on the number of elements in the set A_i^*. It also depends on the resolution with which the input component X_i is mapped into the memory location X_{iq}. For each component in the input vector S the Hamming input-space distance is defined as follows

$$H(X_i)_{12} = \mid X_{1i} - X_{2i} \mid \tag{2.4}$$

There is a value $H_{i_{zero\ overlap}}$ for the Hamming input-space distance where the amount of overlap $\mid A_{i,1}^* \wedge A_{i,2}^* \mid$ between $X_{i,1}$ and $X_{i,2}$ becomes zeros. This value is a function of $| A_i^* |$ and the quantization function $f_i(X_i)$. Thus,

$$H_{i_{zero\ overlap}} = H_{i_{zero\ overlap}}(| A_i^* |, f_i(X_i)) \tag{2.5}$$

Now it is possible to give each component X_i of the input vector a different importance to the output vector $\mathcal{P}$. If $X_i \to X_{iq}$ is a high resolution mapping only a small change in X_i is required to address one or more different elements in A_i^* and to compute a different output vector $\mathcal{P}$. A low resolution mapping $X_j \to X_{jq}$ a large change in X_j is required to address any different element in A_j^* and to compute a different output vector $\mathcal{P}$. The resolution of the mapping is defined in the quantization function $f_i(X_i)$.

It is also possible to use a nonlinear quantization function, as shown in Figure 2.4. This means that high resolution mapping of X_i is used in some

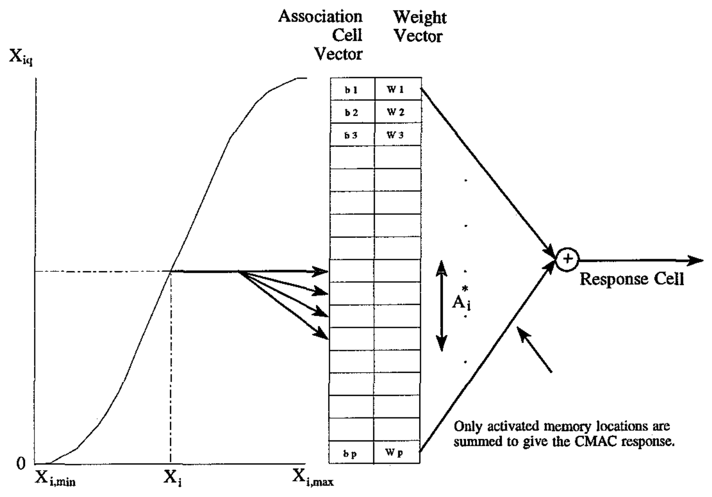

Figure 2.4: Nonlinear Quantization of the input component X_i

parts of the range along the X_i axis and low resolution mapping in other parts of the same axis.

2.4 Data Storage

An important step in the CMAC system is the data storage [5, 4]. The data storage system uses an input scalar $\mathcal{Z}$ to update the activated memory locations. This input scalar $\mathcal{Z}$ depends on the total control system, i.e. whether the CMAC algorithm is applied (i) on-line or (ii) off-line. The objective function of the learning algorithm is the same for these two situations.

The learning rules, used in the learning cycle, are the mathematical representation of the objective function. The learning cycle is repeated until a desired accuracy, the difference between a reference response and the CMAC response, is reached.

The CMAC algorithm is added in parallel with a conventional controller, i.e. on-line as shown in Figure 2.5. The objective function of the on-line

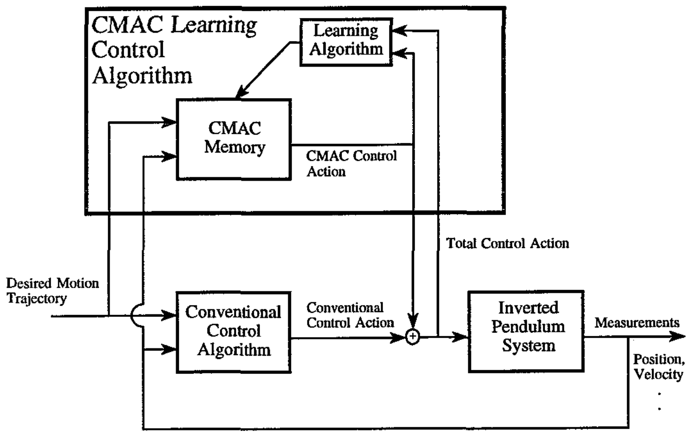

Figure 2.5: The CMAC algorithm added on-line

learning algorithm is to update the activated memory locations in the direction to minimize the difference between the total control action and the CMAC control action. The learning algorithm is as follows,

1. Calculate the CMAC control response,

$$u_{cmac} = \sum_{i=1}^{|A^*|} w_i \tag{2.6}$$

   where w_i is the content of the activate memory locations addressed by the current input vector $\mathcal{S}$.

2. Compare the total control response from the CMAC and conventional controllers, u_{total}, with u_{cmac} and update the content of the active memory locations by

$$\Delta w_i = \beta * (u_{total} - u_{cmac}) = \beta * u_{conventional} \tag{2.7}$$

   where β is the learning rate.

3. Repeat until the required accuracy criteria is reached.

The objective function can only be solved if the conventional controller uses errors, such as position and velocity errors, as input (e.g. PID controller). In other words, the only way the learning stops is when the total control action is equal to the CMAC control action. The action of the conventional controller is then zero, i.e. perfect tracking is achieved. In practical applications the action of the conventional controller will never be zero due to the existing noise level in the system. The required accuracy criteria is then this noise level.

In the off-line configuration, as shown in Figure 2.6, any controller can be used during the learning cycle of the CMAC algorithm. The learning al-

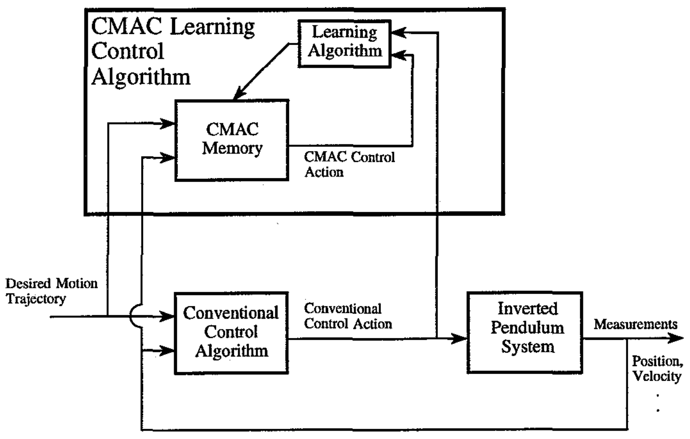
CMAC Learning Control Algorithm
Learning Algorithm
CMAC Memory
CMAC Control Action
Desired Motion Trajectory
Conventional Control Algorithm
Conventional Control Action
Inverted Pendulum System
Measurements
Position, Velocity

Figure 2.6: The CMAC algorithm added off-line

gorithm uses data sets of input and output vectors gained from experiments with the conventional controller. The objective function is to update the activated memory locations in the direction that the difference between the applied control action from the conventional controller and the CMAC control action is minimized. So, the adaptation is the same as for the on-line application with the remark that in Equation (2.5) $u_{total} = u_{conventional}$.

Chapter 3

Experimental

3.1 The Inverted Pendulum

The CMAC algorithm is tested with an experimental mechanical system, i.e. the inverted pendulum as shown in Figure 3.1. The pendulum is connected

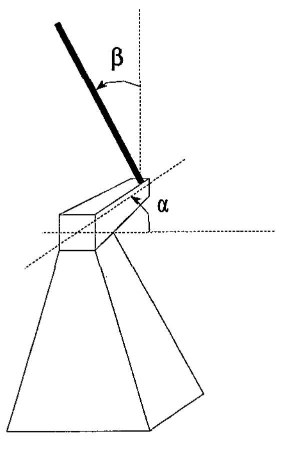

Figure 3.1: The inverted pendulum

with a computer by several interfaces. The computer receives sampled measurements of the angles α and β that can be used for control algorithms. With these measurements and control algorithms the computer calculates the torque to be applied to the inverted pendulum. A motor drives the inverted pendulum with the computed torque. The on-line computations of the control algorithms are performed by a C-coded programm. This programm contains the control algorithm with the measurements as input and the computed torque as output. Results of the experiment are summarized in Matlab format.

3.2 Measurements

The measurements taken of the inverted pendulum are

- the position of the motor, α [rad]
- the position of the pendulum, β [rad]

as shown in Figure 3.1. The resolution of the two encoders is $\frac{2\pi}{4096} = 1.5 * 10^{-3}$ [rad]. The resoultion of the motor angle is 8 times smaller, because of a transmission between the encoder and the actual motor angle α. The resolution of the motor is $2 * 10^{-4}$ [rad]. These two measurements are the input of the C-coded programm.

The positions are measured at discrete equally spread moments, i.e. the sampling time Δt [s]. The sampling time is constant and the sampling frequency is defined as $f_s = 1/\Delta t$ [Hz] (see Figure 3.2). During the sampling

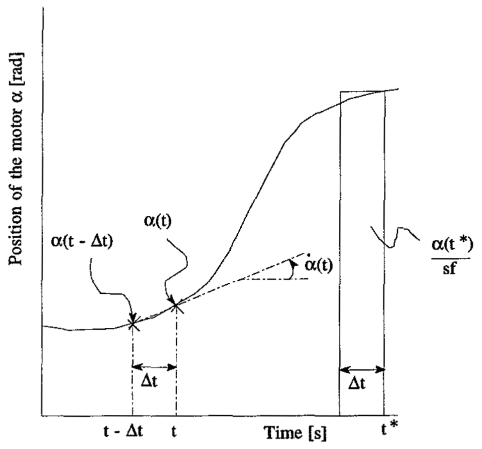

Figure 3.2: The sampling time Δt

time the programm has to compute the torque as output. This torque is then sent to the motor and applied to the inverted pendulum in the same sampling time.

The control algorithm may need more data such as

- the velocity of the motor, $\dot{\alpha}$ [rad/s]

- the velocity of the pendulum, $\dot{\beta}$ [rad/s].

Instead of measuring the velocities these are derived by differentiation of the position measurements (see Equation 3.1 and 3.2 and Figure 3.2).

$$\dot{\alpha}(t) = [\alpha(t) - \alpha(t - \Delta t)] * f_s \tag{3.1}$$

$$\dot{\beta}(t) = [\beta(t) - \beta(t - \Delta t)] * f_s \tag{3.2}$$

This differentiation of the position measurements leeds to a resolution of $f_s * 10^{-4}$ [rad/s] for the velocity. So, if $f_s = 200$ [Hz] the resolution of the velocity is $2 * 10^{-2}$ [rad/s]. The resolution of the velocvity is less accurate then the resolution of the position.

The integral of the positions may also be needed by the control algorithm. These are

- the integral of the position of the motor, $\int\alpha$ [rad s]
- the integral of the position of the pendulum, $\int\beta$ [rad s].

The integrals of the positions are also derived in the C-coded programm as followed

$$\int_0^t \alpha(t) = \int_0^{t-\Delta t} \alpha(t) + \frac{\alpha(t)}{f_s} \tag{3.3}$$

$$\int_0^t \beta(t) = \int_0^{t-\Delta t} \beta(t) + \frac{\beta(t)}{f_s} \tag{3.4}$$

The last term $\frac{\alpha(t)}{f_s}$ in Equation 3.3 is represented in Figure 3.2.

3.3 The Desired Accuracy

The desired accuracy is limited to the resolution of the measurements. Thus, the desired accuracies are as followed

- Position accuracy is $2 * 10^{-4}$ [rad]
- Velocity accuracy is $f_s * 2 * 10^{-4}$ [rad/s]

3.4 Friction

The dynamic system of the inverted pendulum is nonlinear due to static and kinetic friction in the motor. The static friction is $\pm$ 16 [Nm] and the kinetic friction is a function of the velocity of the motor, $\dot{\alpha}$.

3.5 The Quantization Functions

As mentioned before, there are two different quantization functions to be distinguished i.e., (i) linear quantization functions and (ii) nonlinear quantization functions. For the experiments, nonlinear sigmoïde quantization functions are used. These sigmoïde quantization functions are defined as

$$X_{iq} = \frac{\eta}{1 + e^{-\gamma(X_i - \mu)}} \quad with \ \eta = \mid A_i \mid \tag{3.5}$$

In Figure 3.3 a sigmoïde quantization function is shown. This nonlinear

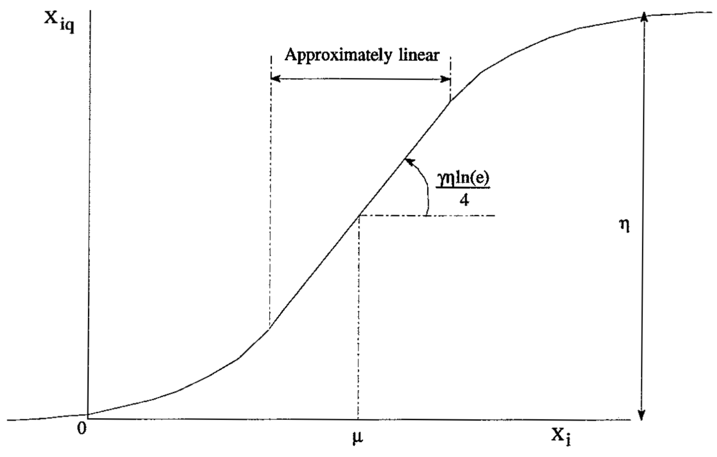

Figure 3.3: The sigmoïde quantization function

quantization function can also be used as a linear quantization function because the sigmoïde function is approximately linear for a certain range of X_i (see Figure 3.3). Another advantage of using these functions is the certainty that X_{iq} will be in the range of $\mid A_i \mid$.

3.6 Available Memory

The number of available memory locations are $\mid A \mid= 8000$. This number is fixed due to the use of PC-Matlab which only allows matrices with a length of $\pm$ 8000 cells. The use of Matlab386, with larger memory facilities, will introduce a memory problem in the C-coded programm if the matrices are larger than 10000 cells.

Chapter 4

Results and Discussion

There are two kinds of experiments tested on the inverted pendulum, as described in section 2.4 'Data Storage'.

- The CMAC algorithm used on-line in parallel with a PID controller. The controller is tested as a friction compensation controller.
- An off-line experiment to test whether the CMAC algorithm can copy a conventional controller for a certain trajectory of the inverted pendulum.

4.1 On-line

This experiment is to test the CMAC algorithm as a friction compensator. To determine the effectiveness of this controller only the motor has to be tested, because the friction is due to the motor and not to the pendulum. Hence, only the position and velocity of the motor have to be measured. The control configuration of the CMAC algorithm and PID controller is given in Figure 4.1 with

$$\underline{y}_{sp} = \begin{bmatrix} \alpha_{desired}(t) \\ \dot{\alpha}_{desired}(t) \\ \int \alpha_{desired}(t) \end{bmatrix} \tag{4.1}$$

$$\underline{y}_{m} = \begin{bmatrix} \alpha(t) \end{bmatrix} \tag{4.2}$$

In the PID control algorithm and the CMAC algorithm, the velocity and the position integral of the motor are derived as described in section 3.2.

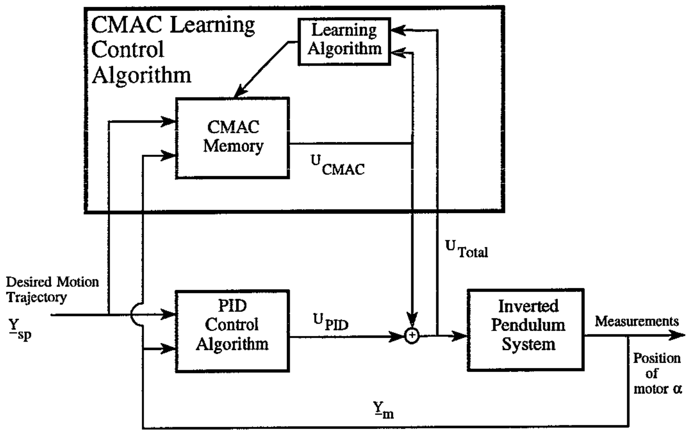

Figure 4.1: On-line control configuration

4.1.1 The Control Algorithms

- The PID controller is defined as follows

$$u_{pid} = \begin{bmatrix} \epsilon(t) \\ \int \epsilon(t) \\ \dot{\epsilon}(t) \end{bmatrix} . \begin{bmatrix} P & I & D \end{bmatrix} \tag{4.3}$$

where P is the Proportional action, I the Integral action and D the Derivative action and

$$\epsilon(t) = \alpha_{desired}(t) - \alpha(t). \tag{4.4}$$

- In the CMAC algorithm, the input vector $\mathcal{S}$ and the output vector $\mathcal{P}$ are

$$\mathcal{S} = \begin{bmatrix} \alpha(t) \\ \dot{\alpha}(t) \\ \int \alpha(t) \\ \epsilon(t) \\ \dot{\epsilon}(t) \\ \int \epsilon(t) \end{bmatrix} \tag{4.5}$$

$$\mathcal{P} = u_{cmac} \tag{4.6}$$

Besides the velocity and velocity error of the motor which are characteristic for the friction, the input vector contains the measurements of the position, position error, integral and integral error of the motor.

- The learning algorithm represents the learning rule which is defined as

$$\Delta w_i = \beta * (u_{total} - u_{cmac}) = \beta * u_{pid} \tag{4.7}$$

4.1.2 The Desired Trajectory

There are two different trajectories tested with each a constant desired velocity of the motor $\dot{\alpha}_{desired}$,

- $\dot{\alpha}_{desired} = 0.5\ [rad/s]$
- $\dot{\alpha}_{desired} = 0.3\ [rad/s]$

This means for the vector $\underline{y}_{sp}$ which specifies the desired motion

- $$\underline{y}_{sp} = \begin{bmatrix} 0.5 * t \\ 0.5 \\ 0.25 * t^2 \end{bmatrix} \tag{4.8}$$

- $$\underline{y}_{sp} = \begin{bmatrix} 0.3 * t \\ 0.3 \\ 0.15 * t^2 \end{bmatrix} \tag{4.9}$$

The desired motions are represented in Figure 4.2 and Figure 4.3.

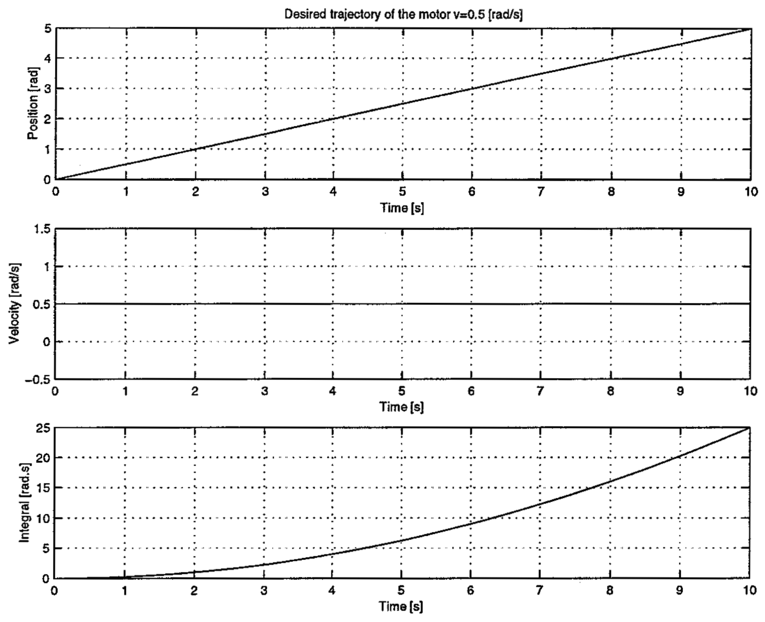

Figure 4.2: The desired trajectory of the motor with $\dot{\alpha}_{desired} = 0.5$ [rad/s]

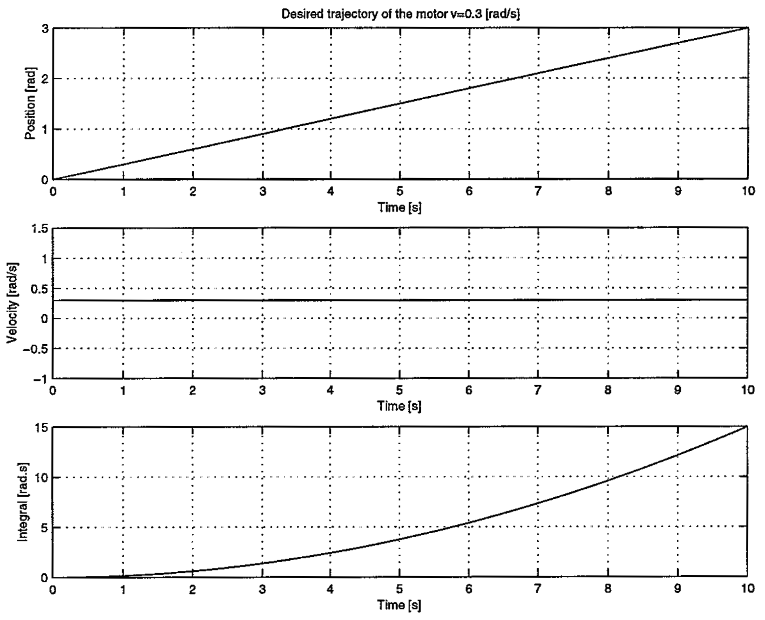

Figure 4.3: The desired trajectory of the motor with $\dot{\alpha}_{desired} = 0.3$ [rad/s]

4.1.3 The Control Variables

There are six control variables which have to be determined. For every control variable there will be an explanation and discussion.

1. The sampling frequency f_s in [Hz] should be chosen large enough, so the discretization error is small. A small discretization error is required for the CMAC algorithm which is designed for continuous control functions. On the other hand the sampling time is limited to the time needed for each computational cycle. The applied sampling frequency is 200 [Hz]. So,

$$f_s = 200 \; [Hz] \tag{4.10}$$

$$\Delta t = \frac{1}{200} = 0.005 \; [s] \tag{4.11}$$

2. The learning rate β should not be chosen very large. This will introduce instability of the total system during training. This means that the weights in the CMAC memory will not converge to stable values. The learning rate β should be small, i.e.,

$$\beta \ll \frac{1}{\mid A^* \mid} \tag{4.12}$$

   If the error of the CMAC response u_{pid} is updated completely over the activated memory locations $\mid A^* \mid$ in one sampling time , the memory has no time to generalize from one learning experience to another. The experimental learning rate used $\beta = 10^{-4}$.

3. The number of learning cycles, which is the number of repeated desired trajectories of the manipulator during the experiment, is related to the learning rate β. If the learning rate β is very small, the CMAC memory is updated with a very small amount of the response error u_{pid}. So, there are a large number of learning cycles needed to update the memory correctly. The smaller β the larger the number of learning cycles which are needed to achieve a desired accuracy. With $\beta = 10^{-4}$, the number of learning cycles to reach the desired accuracy discussed in section 3.3 is $\pm$ 20 cycles.

4. As mentioned before the total generalization size A^* is equally shared over the components of the input vector $\mathcal{S}$. So, each component has

the same generalization size A_i^*.
The desired motion with $\dot{\alpha}_{desired} = 0.5$ [rad/s] is tested with two different generalization sizes $A_i^* = 15$ and $A_i^* = 25$. The other desired motion is only tested with generalization size $A_i^* = 15$.

5. If the generalization size is fixed, like here, the quantization functions have to be determined. There are for each quantization function three parameters which have to be specified (see Figure 3.3).
System knowledge can be used in chosing the parameters. In this application the velocity and the velocity error are very important due to the friction. The quantization functions for these two input components should be high resolution mappings. The other components are mapped with the approximately linear part of the sigmoïde quantization function and with a lower resolution. The parameters are determined on trial and error basis. It is very difficult to find the right parameters. For the different experiments, different parameters are found which are given in Appendix A Table A.1-A.3. In these tables i is the index of the input vector component X_i.

6. The parameters of the PID control action (in Equation 4.3) in the on-line configuration are equal for the experiments with the two different desired motions. The total dynamic system becomes instable

$\dot{\alpha}_{desired}$	P	I	D
0.5	100	3	5
0.3	100	3	5

Table 4.1: The on-line PID parameters for the two desired motions.

if the on-line PID parameters are chosen larger. The same parameters are chosen for an experiment with only a PID control action (so no CMAC algorithm parallel to the PID controller). The results of the on-line experiment are compared with the results given by the experiment with this PID controller. This comparison is to determine whether the CMAC algorithm copies the PID controller.

The results of the on-line experiment are also compared with a PID controller with optimal PID control parameters. For the desired motion with $\dot{\alpha}_{desired} = 0.5$ [rad/s] the optimal parameters are given in Table 4.2. This comparison is to give an impression of the results of the CMAC algorithm used as a friction compensation controller.

$\dot{\alpha}_{desired}$	P	I	D
0.5	2000	1000	35

Table 4.2: The optimal PID parameters for one desired motion.

For simplicity, the different experiments are labeled in Table 4.3 with the different applications and parameters.

Experiment	$\dot{\alpha}_{desired}$	P	I	D	CMAC alg.	A_i^*
1	0.5	100	3	5	Yes	15
2	0.5	100	3	5	No	
3	0.5	2000	1000	35	No	
4	0.5	100	3	5	Yes	25
5	0.3	100	3	5	Yes	15

Table 4.3: The different experiments given in a table.

4.1.4 The Results and Discussion

Comparing Experiment 1 (Figure 4.4) with Experiment 2 (Figure 4.5) it is clear that the on-line configuration of the CMAC algorithm does not copy the PID controller. The CMAC algorithm uses the PID controller as a part

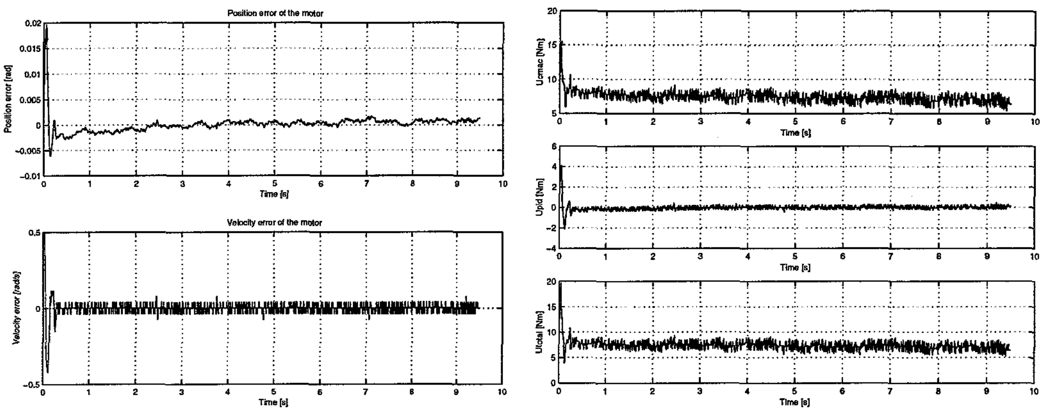

Figure 4.4: The position and velocity errors and the applied torque of Experiment 1.

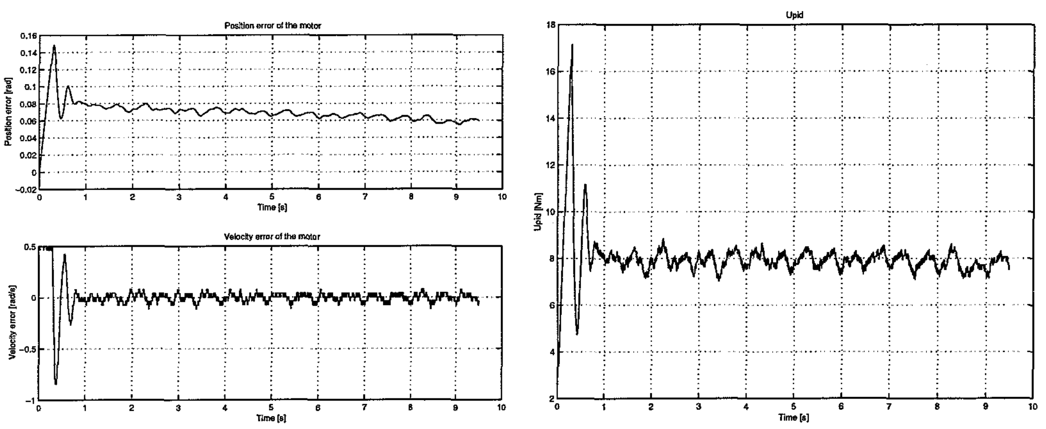

Figure 4.5: The position and velocity errors and the applied torque of Experiment 2.

of the learning algorithm to determine how well the control objective (desired trajectory) is met. The PID control action u_{pid} is nearly zero. Due to the bad accuracy of the velocity of the motor, which is characteristic for the friction, the velocity error is in a range of $\pm$ 0.4 [rad/s]. So, the position error is in a range of $\pm$ 0.002 [rad]. A better result would be found if the velocity was measured and not computed or computed with a higher accuracy.

Experiment 3 (Figure 4.6) compared with Experiment 1 gives an idea of the use of the CMAC algorithm as a friction compensation controller. The

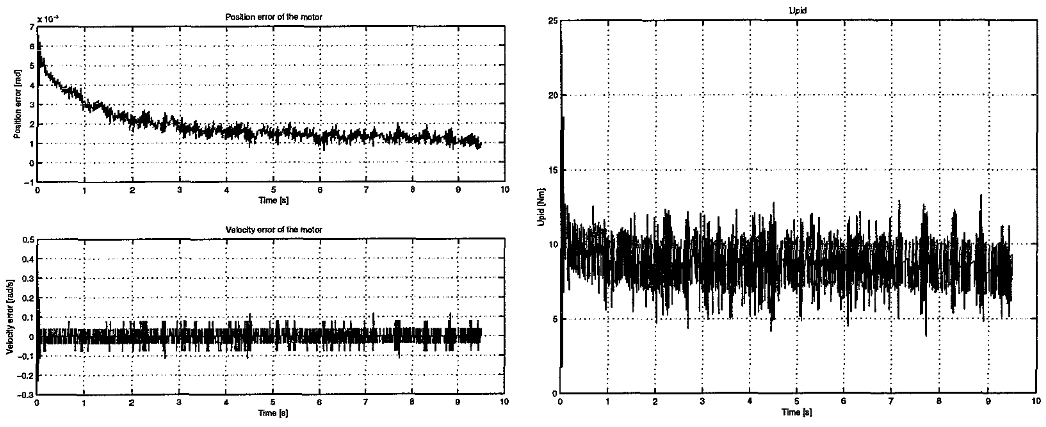

Figure 4.6: The position and velocity errors and the applied torque of Experiment 3.

optimal PID controller has almost the same results as the on-line controller with the CMAC algorithm. The tracking of the desired velocity in Experiment 3 is less accurate than in Experiment 1. The position error is slower in reaching the end accuracy which is approximately equal. The CMAC algorithm is not much better than the PID controller, but with better data of the velocity of the motor the CMAC algorithm might give more accurate results.

Experiment 4 (Figure 4.7) is the same experiment as Experiment 1, but with another generalization size A_i^*. The given $A_i^* = 25$ leads to another set of quantization functions given in Table A.2. The search for the correct quantization parameters is very difficult. It is possible with another generalization size to find the right parameters which lead to the same results as in Experiment 1. The position error in Experiment 4 is smaller than in Experiment 1. The quantization functions are probably chosen better. The difference between the two experiments can be explained with the input-space neighbourhood. As described in section 2.3 there is for each component in the input vector S a value for the Hamming input-space distance where there is no overlap. For the two different experiments these values are given in Table 4.4 for the linear part of the quantization function. The input-space neighbourhoods for the generalization size $A_i^* = 25$ is larger than for $A_i^* = 15$. This does not mean that creating the same input-space neighbourhoods for generalization size $A_i^* = 15$ will lead to the same results. The results are influenced by both the generalization size and the quantization functions.

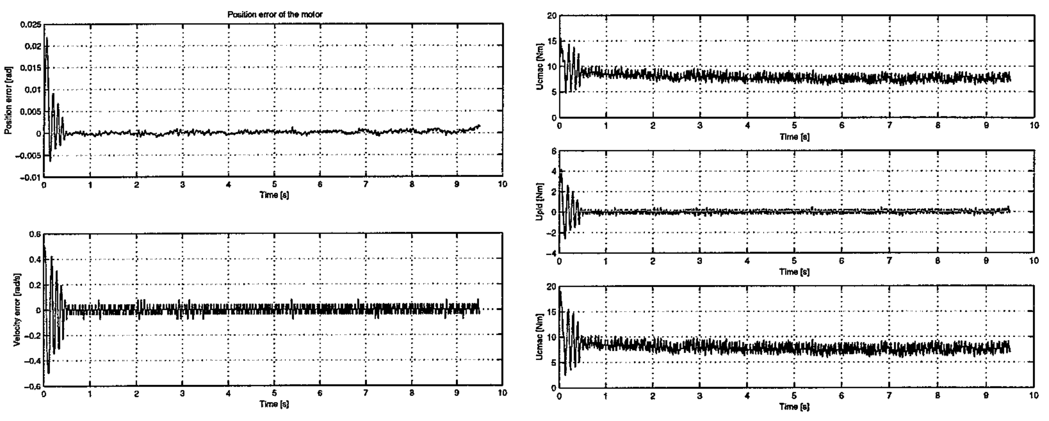

Figure 4.7: The position and velocity errors and the applied torque of Experiment 4.

$H_{i_{zero\ overlap}}$		
X_i	$A_i^* = 15$	$A_i^* = 25$
α	0.00465	0.0076
$\dot{\alpha}$	0.04	0.15
$\int \alpha$	0.0037	0.0114
ϵ	0.0058	0.009
$\dot{\epsilon}$	0.015	0.0255
$\int \epsilon$	0.023	0.038

Table 4.4: The input-space neighbourhoods $H_{i_{zero\ overlap}}$.

This introduces another difficulty in finding the optimal CMAC parameters. The generalization size can not be chosen freely and than find the best set of quantization functions. It is a combination of these two property which specify the results of the CMAC algorithm.

Experiment 5 (Figure 4.8) is to test smaller desired velocities of the motor. The desired velocity of 0.3 [rad/s] is the smallest velocity where the CMAC algorithm still works well without larger velocity errors of the motor. Smaller

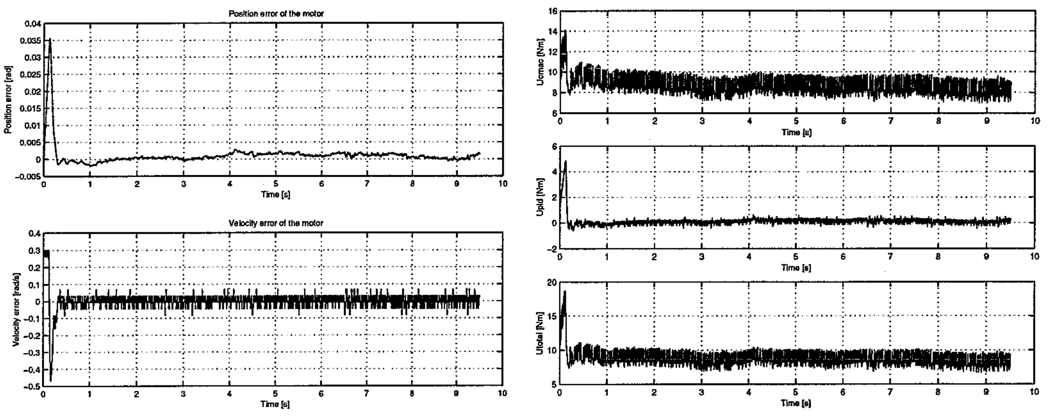

Figure 4.8: The position and velocity errors and the applied torque of Experiment 5.

velocities of the desired trajectory makes the dynamic system instable during training. This is also due to the inaccurate data of the velocity.

4.2 Off-line

This experiment is to test whether the CMAC algorithm can copy a conventional controller for a certain trajectory of the inverted pendulum. The off-line configuration is given in Figure 2.6. The conventional controller is a Computed Reference Computed Torque Controller, which keeps the pendulum up and the motor rotates.
In the CRCTC algorithm and the CMAC algorithm, the velocity and the position integral of the motor are derived as described in section 3.2.

4.2.1 The Control Algorithms

In the CMAC algorithm, the input vector $\mathcal{S}$ and the output vector $\mathcal{P}$ are

$$\mathcal{S} = \begin{bmatrix} \dot{\alpha}(t) \\ \beta_{desired}(t) - \beta(t) \\ \beta(t) \\ \dot{\beta}(t) \end{bmatrix} \tag{4.13}$$

$$\mathcal{P} = u_{cmac} \tag{4.14}$$

With β is the angle of the pendulum. The choice of this input vector is free, but it is clear that it is important to have data of the position and velocity of the pendulum to keep the pendulum up.

4.2.2 The Desired Control Function

The desired control function performed by the CRCTC is represented in Figure 4.9. During training different datasets are used. These are necessary for

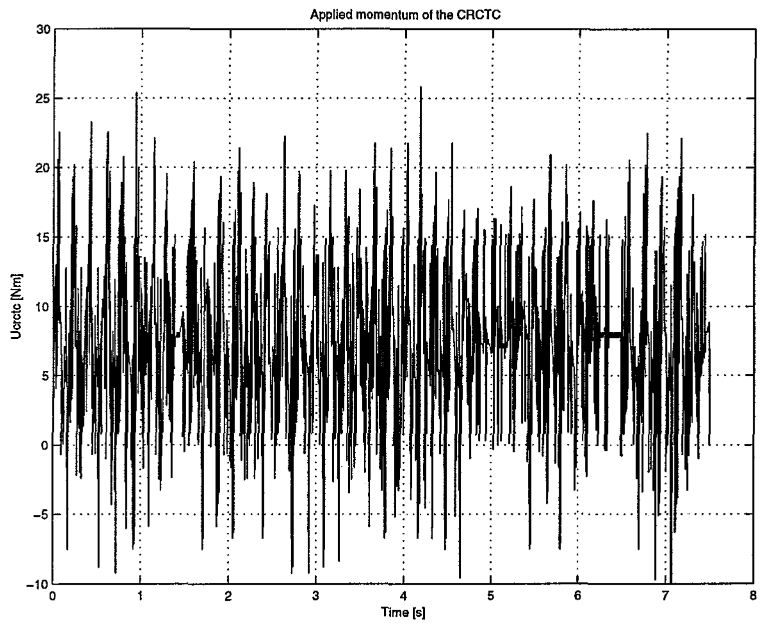

Figure 4.9: The desired control function of the off-line experiment.

the learning of different input states during the trajectory of the pendulum.

4.2.3 The Control Variables

The control variables are the same as those of the on-line configuration, only the quantization parameters are different.

4.2.4 The Results and Discussion

The results are given in Figure 4.10. The CMAC response is almost the same as the CRCTC response. The peaks Figure 4.10 are untrained input states $\mathcal{S}$ for the CMAC algorithm. It is important to have enough different datasets which are collected from experiments with the CRCTC.

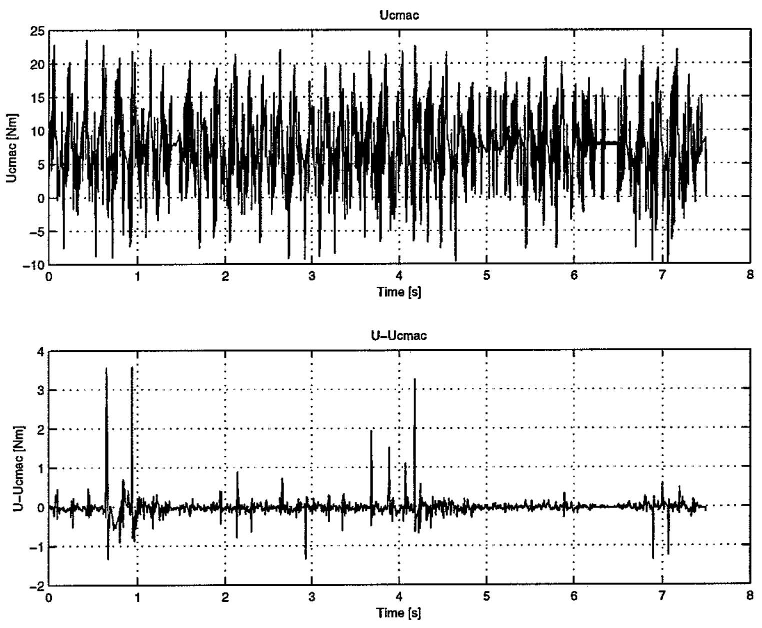

Figure 4.10: The CMAC response and the response error.

Chapter 5

Conclusions

The CMAC algorithm can be used in two applications (i) the on-line configuration and (ii) the off-line configuration.

In the on-line configuration the CMAC algorithm needs an input which specifies how well the control objectives are met. Such an input could be a PID controller which action is zero if there is perfect tracking, so the control objective is perfectly met. The CMAC algorithm does not copy the PID controller but uses it only during training. Nonlinearities, such as friction, which can not be controlled appropriately with a PID controller are no difficulty for the CMAC algorithm. The on-line configuration can also face nonstationary nonlinearities which vary slowly through time.

The CMAC algorithm in the off-line configuration is able to copy a control action for a desired trajectory performed by another controller. The CMAC algorithm will only copy that controller for that trajectory. The performance of the CMAC algorithm is at best the same as that of the original controller.

During the learning cycle the objective is to optimize the performance of the CMAC memory. The performance after training is completely limited by one of the following:

- insufficient sensor input data
- insufficient mapping resolution of the various quantization functions
- insufficient speed in the CMAC computational cycle
- insufficient memory locations

The mapping resolution of the quantization functions is an important part of the CMAC algorithm. The quantization functions and generalization size specify the CMAC algorithm. Though the right parameters of the quantization functions and generalization size are difficult to find and based on trial and error.

Chapter 6

Recommendations

For the study of the CMAC algorithm as a friction compensation controller another application has to be tested with more accurate velocity data. This data can be more accurate if the velocity is measured on-line or the velocity is computed more precise. The CMAC algorithm can then be compared with other friction compensation controllers and be classified.

The difficulty of finding the right CMAC parameters could be overcome if these parameters, quantization function and generalization size, are adapted on-line during training by learning rules. These learning rules should adapt the parameters in that direction that the PID action is minimalized. So,

$$\frac{\partial u_{pid}}{\partial \gamma} < 0 \tag{6.1}$$

$$\frac{\partial u_{pid}}{\partial \mu} < 0 \tag{6.2}$$

$$\frac{\partial u_{pid}}{\partial \mid A_i^* \mid} < 0 \tag{6.3}$$

A study of these learning rules could make the CMAC algorithm easier to use.

During training the dynamic sytem becomes often instable. A study of the stability of the CMAC algorithm in relation with the learning rules and CMAC parameters could give more sight in these problems.

Appendix A

The Quantization Functions

i	γ	μ	η
1	0.1	0	1333
2	1	0	1333
3	0.08	0	1333
4	8	0	1333
5	3	0	1333
6	2	0	1333

Table A.1: The quantization functions for $\dot{\alpha}_{desired} = 0.5$ [rad/s] $A_i^* = 15$

i	γ	μ	η
1	0.3	0	1333
2	2	0.3	1333
3	0.15	0	1333
4	8	0	1333
5	3	0	1333
6	2	0	1333

Table A.2: The quantization functions for $\dot{\alpha}_{desired} = 0.5$ [rad/s] $A_i^* = 25$

i	γ	μ	η
1	0.4	0	1333
2	1.2	0	1333
3	0.17	0	1333
4	8	0	1333
5	3	0	1333
6	2	0	1333

Table A.3: The quantization functions for $\dot{\alpha}_{desired} = 0.3$ [rad/s] $A_i^* = 15$

Bibliography

[1] J. S. Albus. A Theory of Cerebellar Function. *Mathematical Biosciences*, Vol. X:pp. 25–61, 1971.

[2] J. S. Albus. A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC). *ASME Journal of Dynamic Systems Measurements and Control*, pages 220–227, 1975a.

[3] D. Knuth. *The Art of Computer Programming*, volume 3 of *Sorting and Searching*. Addison Wesely, 1973.

[4] G. A. Larsen, S. Cetinkunt, and A. Donmez. CMAC Neural Network Control for High Precision Motion Control in the Presence of Large Friction. *ASME Journal of Dynamic Systems Measurements and Control*, Vol. 117:pp. 415–420, September 1995.

[5] J. S. Albus. Data Storage in the Cerebellar Model Articulation Controller (CMAC). *ASME Journal of Dynamic Systems Measurements and Control*, pages 228–233, 1975b.