DESIGNING TEACHABLE ROBOTS

A Thesis

presented for the degree of

Doctor of Philosophy

in Electrical and Electronic Engineering

in the

University of Canterbury,

Christchurch, New Zealand

by

Bruce A MacDonald BE (Hons)

University of Canterbury

1984

CONTENTS

LIST OF FIGURES

Acknowledgements

I am grateful to my supervisor, John Andreae, for his excellent guidance and support. John always had time to talk to me. He always took the time to carefully read and constructively criticise my work, returning it quickly. John gave me unrestricted use of his computer, both for my work and for producing most of this thesis. I thank Molly Andreae for numerous meals and cups of coffee.

I am grateful to the University Grants Committee for the scholarship which supported me. The Electrical and Electronic Engineering Department supported my work. Those who attended the Language and Thought meetings run by John Andreae and Kon Kuiper provided me with an important forum. The General Manager of New Zealand Electricity granted me leave without pay and provided facilities for producing some of this thesis. Berenice Crowther typed three large chapters.

My family supported me throughout.

Finally, my wife, Sue Stodart, gave me continuous support and encouragement, and helped in the thesis production.

Author's Note

Although this thesis is long, about 480 pages, the main text is only about 250 pages. Appendices are at the end of each chapter. The 80 pages of the chapter V appendix contain the details and discussion of a very simple demonstration "arm-robot", and so are not essential to the main thesis argument, but are necessary for the thesis to be complete.

Chapters II, VI to IX and appendix A at the end of the thesis were printed on a slightly different machine from the rest of the thesis.

ABSTRACT

This thesis advances the design of teachable adaptable robots. I propose two paths of improvement to the popular, easy to use, leading method, in which a teacher literally leads the robot by its hand through movements. The improvements enable motor commands to be changed and conditional branches to be formed, without the need for a keyboard or other explicit programming device. On improvement path 1, the addition of a verbal correcting (VC) scheme would enable a teacher to make on-line verbal corrections to a robot's movement sequences. The further addition of a production system of corrections (PSC) would enable a robot to remember and use verbally taught conditional corrections. On path 2 a goal-seeking (GS) system and VC would enable a teacher to set goals, lead movements, and verbally correct the robot. The robot then selects its own motor commands for achieving goals.

A multiple context learning system (MCLS), a multiple, extended GS system, combines the two paths. It enables both sequences and goals to be taught to a led robot. A simple, but real, led MCLS-robot is demonstrated.

I establish four important properties of MCLSs: (a) an MCLS can enable a robot to learn to perform motor commands that are initially performed only by reflex, so that eye and speech motor commands, neither suitable for being led, can still be learned; (b) an MCLS can learn to be a Turing machine, which is a universal computing machine, explicitly showing the error in criticisms of MCLSs' computational power; (c) the selections of a context learning system in an MCLS converge on the optimal motor commands for achieving goals; and (d) an MCLS-robot can handle the negation problem; doing something positive in the absence of a certain condition.

CHAPTER I

INTRODUCTION

This thesis advances the design of teachable adaptable robots. Robot internal representations are developed along with the teacher-robot interaction. The result is that robots would be easier to teach, and could learn more difficult and a wider variety of tasks; they would be more teachable. Also, robots could learn to cope more quickly with a wider variety of situations. In learning to cope with different situations robots would require less teacher assistance. They would be more adaptable, as well as more teachable. I will explain that existing robots are both (a) difficult to teach, and (b) not very adaptable.

The teaching method that I will mainly be concerned with is the lead-through, or leading method. The teacher teaches a robot to perform a movement, by moving the robot's arm through that movement while the robot is in a training mode. The robot remembers the movement, and may execute it later while in its execution mode.

Now, when using popular leading a teacher must explicitly program the robot via a keyboard-type device, if either motor commands must be changed or conditional branches formed. The advances I propose remove this need for a keyboard-type device, but retain two important aspects of leading:

(a) the teacher need have no expertise in explicit programming; and

(b) the teaching will be most suited to a teacher who is expert at the task to be taught.

I will propose two paths of improvement to the popular

implementation of the leading method. On the first path, the teachability of led robots is increased by the addition of an on-line verbal correcting (VC) scheme. The scheme enables a teacher to give verbal corrections for external disturbing forces while the robot executes a sequence of movements. Thus tasks requiring corrections to be made for external forces can be easily taught. Also on the first path the teachability and adaptability of the leading method is increased by the further addition of a production system of corrections (PSC). A PSC would enable a robot to remember and use the experience of correcting given by VC. The robot remembers which correction to make under which conditions.

On the second path of improvement, a goal-seeking (GS) system is employed with leading. It enables the robot to work out its own motor command sequences for achieving teacher set goals. The teacher would lead individual motor commands, set goals and, if VC is added to the GS system, make verbal corrections. The GS system increases the teachability by enabling goals and subgoals to be taught. The adaptability is also increased because the robot can select its own motor command sequences for achieving a goal or goals.

Now, path 1 improvements cannot enable goals to be learned, but only motor command sequences. Path 2 improvements cannot enable motor command sequences to be learned, but only goals. Having established the two improvement paths, I go on to show how a multiple, extended GS system, called a multiple context learning system (MCLS), can enable a led robot to both perform sequences and achieve goals. I report a simple demonstration of a led robot with an MCLS. MCLSs are being developed by John Andreae and his

coworkers for intelligent robots of the future. References are given in section 2.

I then go on to establish four other properties of MCLSs. Firstly, leading is not suitable for teaching eye and speech movements. So I show that a robot with an MCLS in it can learn to perform eye motor commands from built-in eye reflexes. A few specific eye and speech reflexes can result in general learning of eye and speech movements by an MCLS.

Secondly, I show that the class of MCLSs I use for both leading and reflex learning, has more "power" than some critics think. MCLSs have been criticised for their lack of "power": "... it seems clear that the machine can only learn finite automata. Something of the power of an augmented transition network would be forever beyond it. So it certainly couldn't learn a grammer for any reasonably interesting subset of English." (p.10 Hayes, 1977). It should have already been clear that MCLSs are not subject to this criticism. I explicitly dimiss the criticism by showing how a simple MCLS learns to be a Turing Machine, the theoretical machine with the greatest "power" in the sense used by Hayes.

Thirdly I establish the optimality and convergence of the goal seeking process that is implemented by an MCLS. Finally I describe how a robot with an MCLS can handle the problem of negation; the problem of doing something positive in the absence of a certain condition.

Chapter II explains that existing robots lack teachability and adaptability. Chapter III proposes the two paths of improvement to the popular leading method. Chapter IV explains how an MCLS can both perform sequences of movements and achieve goals. Chapter V

reports a simple demonstration of a led robot with an MCLS. Chapter VI establishes the convergence properties of the goal seeking process in an MCLS. Chapter VII shows that an MCLS can learn to perform actions initially performed only by reflex. Chapter VIII shows how an MCLS can learn to be a Turing Machine. Chapter IX shows how an MCLS can handle the problem of negation in three ways.

Section 1 gives a brief summary of each chapter. Section 2 briefly states the history of the work in the thesis. Section 3 sets out the organization of the thesis.

I.1 CHAPTER SUMMARIES

CHAPTER II EXISTING ROBOTS ARE STUPID

Industrial robots are difficult to teach. Industrial robots are not adaptable, can carry out manual tasks only in a limited range of situations, and lack the ability to use sensory information. In general the environment in which a task is to be performed cannot be completely determined beforehand. Existing industrial robots cannot cope with uncertainty about the environment. Existing robots have only limited abiliy to coordinate their arms with sensory information. Industrial robots lack sensors, so their capacity to respond to conditions in the environment is limited. The human performs so well because of the richness of his sensory feedback, rather than the mechanical efficiency of his arm, which is not high. No existing robot can perform complex movement tasks in the real world, given the sorts of task description we are used to giving.

CHAPTER III ENHANCING THE LEADING METHOD

A teacher must use a keyboard device on existing industrial robots if he wants to (a) correct taught motor commands, or (b) teach conditional branches. This "explicit" programming of corrections and branches is difficult and unnatural; especially if the teacher of the led robot is someone skilled in the task being taught, but not a programmer. I propose two paths of improvement to the popular implementation of leading. Path 1 is the addition of both verbal correcting (VC) and a production system of corrections (PSC). Path 2 is the use of a goal-seeking (GS) system to which VC is added. VC should increase the teachability of led robots by enabling a teacher to verbally correct motor commands on-line, using his own natural ability at verbally correcting humans. A PSC and a GS system should also increase the teachability of led robots. They would enable a teacher to teach conditional branches using his own natural ability at leading and VC, and his own knowledge of the task goals; a PSC for teaching sequences of movements, and a GS system for teaching sequences of goals. Having sensed the environmental conditions, a robot would select its own motor commands, once (a) the motor commands have been led, (b) verbal corrections made, and (c) for a GS system, the goals set. A PSC and a GS system should increase the adaptability of led robots by enabling a robot to select its own motor commands, given the sensed robot conditions.

CHAPTER IV LEADING WITH AN MCLS

An MCLS can enable a robot to perform sequences of movements and achieve goals. A GS system enables only the achievement of goals. A PSC enables only the performing of movement sequences. An MCLS comprises several rules and a decision procedure. The word "rule" is used in two ways in this thesis. When underlined, that is "rule", it specifically means the MCLS variety, while just "rule" has the normal meaning. A rule is similar to a GS system, but proposes motor commands to perform on the basis of the robot's recent history. A GS system performs motor commands on the basis of only the immediate situation. The decision procedure of an MCLS selects motor commands to perform from all the motor commands proposed by all the rules.

CHAPTER V A SIMPLE LED ROBOT WITH AN MCLS

The arm-robot, comprising a real metal lever arm and a simple MCLS, demonstrates how an MCLS can implement leading. The teacher leads the robot's arm into a light beam. The arm cuts the beam, causing a goal to be set for that arm angle. The teacher leads some more movements. The arm-robot then lifts its arm into the light, while there is a weight on the arm that makes the arm sag.

CHAPTER VI OPTIMAL GOAL SEEKING

I show that a successive overrelaxation method, called "leak-back", can be used on a rule in an MCLS, to work out the "best" motor command to perform for seeking a goal. The best motor command may be either one that optimizes the total expected goals reached, or one that minimizes the number of motor commands that must be performed before reaching the next goal. Leak-back operates

simultaneously with the decisions of an MCLS. The MCLS does not wait for leak-back to converge on optimal motor commands. The decisions determined by the current stage of the leak-back process are used by the MCLS.

CHAPTER VII A FRESH LOOK AT CONDITIONED REFLEXES

A teacher will have no natural feeling for the eye motor commands or eye movements required for a task. He would not have the same natural feeling for the visual information required for a task, as he would have for the physical manipulations required. He would have difficulty making a detailed description of the eye movements or motor commands required for a task. Instead of the teacher providing them, eye movements might be preprogrammed to occur in a specific way, when the robot is built. [For example, a robot's eyes might be preprogrammed to follow its hand.] In that case, the robot would have to be able to learn to use the preprogrammed eye movements in situations for which they had not been preprogrammed. I show that a robot with one of the simplest forms of preprogrammed eye movement can learn to perform the movements in situations for which they are not preprogrammed. The simple preprogrammed movements are reflexes; they are simple fixed movements that are triggered by particular visual stimuli. The MCLS PURR-PUSS (Andreae, 1977a; 1979b) is taught to perform a look-left motor command which is initially performed only by reflex.

CHAPTER VIII ANSWERING THE CRITICS

In Hayes's (1977) criticism of MCLSs he said that the MCLS PURR-PUSS "... can only learn finite automata ..." and "So it certainly couldn't learn a grammar for any reasonably interesting subset of English ..." (p. 10). But grammars are finite sets of rules. So there is a finite automaton for every grammar. It has already been shown that an MCLS can learn any finite automaton that will fit into its memory (Andreae & Cleary, 1976). So an MCLS can learn any grammar that will fit into its memory. An MCLS's memory could hold an enormous grammar. Since the importance of an MCLS's being able to learn any finite automaton does not seem to have been recognized by the critics, as exemplified by the passage quoted from Hayes above, I explicitly show that an MCLS can learn any grammar that will fit into its memory. To do this I describe an MCLS that learns to simulate a simple automaton and its "tape" memory. The tape memory is inside the MCLS. The tape can have a description of a grammar on it. The MCLS learns this description. The simulated automaton reads the description and implements the grammar. Another part of the tape is used as working memory, for intermediate results. The automaton is what is called a "universal Turing machine".

CHAPTER IX WHEN IS 'NOT' NOT 'NOT' FOR A ROBOT

It will be important for a robot to be able to do something positive in the absence of a particular condition. For example, on not recognizing something, say a person's face, a robot should do something positive on the basis of not recognizing the face, like introducing itself and saying "Hello". The problem of doing

something positive in the absence of a condition will be called the "negation problem". The absence of the condition must actually be an absence. The absence must not be represented by some stimulus that is present. I show that an MCLS can do something positive in the absence of a condition, in three ways, each by a different MCLS. The first negation handling MCLS (NHM) responds to the absence of a condition only if the actual condition present is one that the MCLS hasn't had before. The second MCLS, NHM', responds to the absence of a condition whether the actual one is new or not. However, NHM' can be taught to respond to the presence of the particular condition with the absence response! NHM", the third MCLS, responds to the absence of a condition whether the actual condition is new or not, and cannot have its presence and absence responses changed once they have been learned.

I.2 HISTORY OF THE WORK IN THE THESIS

Earlier versions of the work in this thesis are given in MacDonald (1979; 1980; 1981; 1982a; 1982b); specifically chapter II (1981), chapter III (1982b), section 1 of chapter IV (1982b), chapter V (1981; 1982a), chapter VI (1982a), section 3 of chapter VII (1982a), the rest of chapter VII (1979), chapter VIII (1980); chapter IX (1980; 1981). Various work on MCLSs, the subject of chapter IV, has been published (Andreae, 1977a; 1972-1983; Andreae & MacDonald, 1981; Andreae & Andreae, 1979; Andreae & Andreae, 1978; Andreae & Cleary, 1976; Cleary, 1980a; 1980b; MacDonald & Andreae, 1981). Some of the work in chapter VIII is also in MacDonald and Andreae (1981).

I.3 ORGANIZATION OF THE THESIS

Chapter II presents the state-of-the-art of robot teachability and adaptability. Chapter III proposes major improvements to led robots. Chapter IV carries the advances on to the MCLS. Chapter V reports an actual demonstration of the proposed advanced MCLS-led-robot design. Chapters VI through IX establish important properties of the design, as set out in section 1 above and briefly summarized in the paragraph before section 1.

A glossary of important words and abbreviations is given at the end of the thesis, just before the thesis appendix. The glossary refers to the text where the entries are explained in more detail. The thesis appendix comprises (a) comments on humans and robot design, and (b) suggested future research.

I.4 SUMMARY

This thesis advances the design of teachable adaptable robots by developing robot internal representations and teacher-robot interactions. Robots would be more teachable and more adaptable.

CHAPTER II

EXISTING ROBOTS ARE STUPID

This chapter is concerned with the teachability and adaptability of existing robots. It explains that existing robots lack both.

All but the simplest robots, for example those with only manually set mechanical stops, have a main controller which sends motor commands to the motor control systems in the robot's body, in order to perform a movement task. This is depicted in Figure II-1. The arm control system is an example of a motor control system. In order to get a robot to perform a movement task, a teacher may

(a) tell or show the robot the motor commands to send to the body

e.g. instructions fed in on a paper tape; or motor commands stored during remote control by the teacher

(b) tell or show the robot the movements and forces to achieve

e.g. programmed movements, say, MOVE X=10,Y=20; or teacher manually leading the robot through movements.

(c) give the robot a task description in more complex terms than the movements and forces required

e.g. English description; or high level program, say PICK_UP BLOCK.

For (b) and (c), the robot must determine the motor commands to send to the body. This determination may be very simple. For example, consider

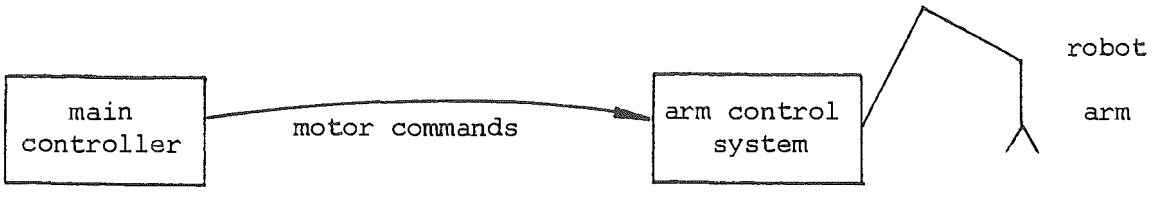


Figure II-1 The main controller in the robot sends motor commands to the motor control systems of the robot's body; for example the arm control system shown in the Figure.

the position error controller, depicted in Figure II-2, for controlling the angular position of a motor shaft. Suppose the sensitivity of the feedback from the shaft position sensor is one volt per degree. To move the shaft to 30 degrees a motor command of 30 volts might be sent to the control system. The determination may also be complex, for example when the dynamics of an arm and load are modelled in order to determine the motor commands required for achieving movements.

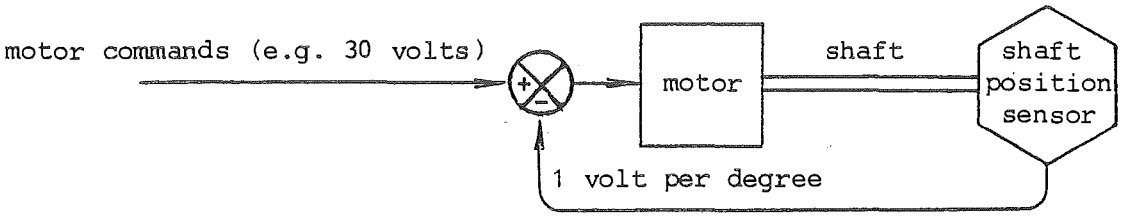


Figure II-2 A simple position error controller for a motor shaft

A robot may acquire some of the ability to determine motor commands from movements and forces, and from a task description. The teacher may be involved in the robot's acquiring of these abilities. For example, a scheme has been described in which a robot is preprogrammed in a high level language, leaving some position transformations unspecified (Takase et al, 1981). An operator may teach these transformations by manually moving the arm during the task. This enables the robot to calculate later movements.

So the teachability of a robot is concerned with the interaction between the robot and the teacher, in (i) giving descriptions, (ii) telling and showing movements and forces, (iii) telling and showing motor commands, and (iv) helping a robot determine motor commands given descriptions, movements and forces. The teachability of a robot is partly a subjective judgement, since it depends on the interaction between the robot and the teacher. The teacher is the one who judges the teachability, but he is not a passive observer.

I expect that human teachers will find robots more teachable

(a) the easier it is to teach tasks,

(b) the greater the range of tasks that can be taught,

(c) the more difficult the tasks that can be taught.

Different teachers may not agree on the relative importance of these three, but most will agree that the greater the range of tasks, the greater the difficulty of tasks, and the easier the teaching, the greater the teachability. It is explained in this chapter that existing robots lack teachability in all three ways.

The adaptability of a robot is to do with how the robot copes with changing environmental conditions. A motor command will produce the required movement or force under some conditions, but not under others. In determining the motor commands required for a task from a description or from movements and forces, a robot may take into account the environmental conditions. Judgements of adaptability are somewhat subjective because different environmental conditions may be seen, by a human observer, as either (a) part of the task, for example different object positions in a bin-picking task, or (b) a change in the conditions for the task, for example, different object orientations for a spray-painting task. That objects will have different positions is implicit in the task of "bin-picking"; but different object orientations are not implied in industrial spray-painting tasks.

I expect humans to find robots more adaptable

(a) the greater the range of conditions they perform a task in,

(b) the less teacher assistance is required for enabling a robot to determine the motor commands required when conditions change,

(c) the less time it takes the robot to return to proper performance of a task when the conditions change.

Different humans may not agree on the relative importance of these three, but most will agree that the greater the range of conditions, the less teacher assistance, and the quicker the robot's adaption, the

greater the adaptability. It is explained in this chapter that existing robots lack adaptability in all three ways.

Section 1 discusses the three ways in which existing robots lack teachability, and the three ways in which existing robots lack adaptability. Section 2 describes the state-of-the-art in the determination of motor commands from given movements and forces, or a given task description. In section 3 teaching methods for existing robots are described and compared. Section 4 describes the state-of-the-art in the determination of motor commands required for different environmental conditions. Section 5 discusses the ways existing robots cope with a range of environmental conditions. Some examples of existing robot systems are discussed briefly in the appendix.

II.1 LACK OF TEACHABILITY AND ADAPTABILITY

II.1.1 Teachability

At the start of this chapter it was suggested that, in interacting with a robot, a human will find the robot more teachable (a) the easier the teaching is for the teacher, (b) the greater the range of tasks that can be taught, and (c) the more difficult the tasks that can be taught.

The sort of things a teacher might do in getting a robot to perform a task are shown in Figure II-3. Items further down the list in Figure II-3 are generally more difficult for a teacher to do. However the list says nothing about the range and difficulty of tasks that can be taught. For example, some teaching methods may enable a teacher to get a robot to exert specific forces in the environment, while other teaching methods may not. Specific forces may be required for tightening a nut on a bolt.

Items further down the list shown in Figure II-4 have less difficult and smaller ranges of task.

The teachability of existing robots is low. On both the list in Figure II-3 and the list in Figure II-4, existing robots are not placed high. Humans abilities are actually well above these lists. They can perform tasks for someone without being told to. They can invent tools and machines, as well as use them. Sections 2 and 3 detail the low teachability of existing robots.

Say a short English description of the task,
e.g. "Paint those cars blue"

Write down a short English description of the task

Say a detailed English description of the task,
e.g. "Paint the two cars on assembly
line 1 with the paint in can 3, ..."

Lead the robot through the movements for a task,
e.g. lead a sequence of painting movements

Guide the robot through the movements for a task

Write a program in a high level programming language,
e.g. PICK_UP BOLT; MOVE_TO BOLT_HOLE;
INSERT BOLT IN BOLT_HOLE

Write down a program for the task in an assembler language

Set up mechanical stops for the robot's movements

Figure II-3. A list of things a teacher might do to get a robot to perform a task. Items further down the list are more difficult to do.

Use tools and machines	e.g. those a fitter & turner uses
Use a particular tool	e.g. spanner
Repeat one of a repertoire of movement sequences, each for different environmental conditions	e.g. assemble a variety of products as they arrive at the robot's workbench
Repeat a specific sequence of movements	e.g. a fixed assembly sequence
Repeat a specific sequence of motor commands	

Figure II-4 Items further down this list have smaller and less difficult ranges of task

II.1.2 Adaptability

At the start of this chapter it was suggested that humans would find robots more adaptable (a) the greater the range of situations a task is performed in, (b) the less teacher assistance is required for enabling a robot to determine the motor commands required when the environmental conditions change, and (c) the less time it takes the robot to adapt when conditions change.

Items further down the list in Figure II-5 represent smaller ranges of situation. Items further down the list in Figure II-6 represent more teaching required for enabling a robot to cope with a change in situation. Items further down the list in Figure II-7 represent slower robot adaption. Gaines (1969) distinguishes different varieties of adaptive behaviour. Briefly, he indicates that the more situations a control system can adapt to, and the quicker it does adapt to them, the more adaptive it is. For robot adaptability, the amount of teaching is also important.

The adaptability of existing robots is low. On the lists in Figures II-5, II-6 and II-7, existing robots are not placed high. Humans' abilities are actually well above these lists. They can adapt to changes of environment, performing tasks in very different ways with different materials. They can remain adapted to a huge variety of situations, over a long period of time. It is reported that industrial robots are not adaptable (Nitzan, 1979), can carry out manual tasks only in a limited range of situations (Benati et al, 1980), and lack the ability to use sensory information (Rosen, 1979). Sections 4 and 5 detail the low adaptability of existing robots.

Adjustment to the environment in general	e.g. move objects out of the way and align an object to be painted
Adjustment to the locations of objects in one's day to day environment	e.g. fetching an object from a normal factory environment.
Adjustment to objects in the immediate vicinity	e.g. avoid obstacles during a painting task
Adjustment to variation in shape and size of an object being manipulated	e.g. bin-picking assorted objects
Adjustment to variation in the alignment of an object being manipulated	e.g. bin-picking several objects the same.

Figure II-5 Items further down this list represent smaller ranges of situation

Say a short English description of what to do when various conditions change	e.g. "Move any obstacles out of the way before you begin painting"
Say a detailed English description of how to do a task under a range of conditions	e.g. "Look for a trolley within 3 m of you. If you find one then approach it, grasp the handle, and move it away to the left ..."
Show extra movements for avoiding obstacles	
Interactively change a high level program	e.g. Park & Burnett's (1979) interactive compiler which, for example, allows a program to continue after it has been stopped and changed
Rewrite the program for the task so that it works under a new set of conditions	

Figure II-6 Items further down this list represent more teaching required for enabling the robot to cope with a change in situation

Robot is immediately adapted to many situations once it has adapted to one situation

Robot adapts to a change in conditions over one or two performances

Robot adapts to a change in several hundred performances
e.g. Raibert's (1978) scheme which enables a robot arm to acquire a tabular dynamic model of its arm and load (see section A.4.4)

Robot doesn't adapt by itself at all

Figure II-7 Items further down this list represent slower robot adaption

II.2 DETERMINATION OF MOTOR COMMANDS

The requirements for the determination of motor commands and the state-of-the-art of satisfying those requirements are given in this section.

In many existing robot systems the determination of motor commands from a description of a task is divided into two parts: (a) given a description of the task, the required movements and forces are determined, and (b) given the movements and forces that are required, the motor commands to send to the robot's body are determined. This division may be convenient if sensory information must be taken into account. For example visual information may be related most easily to the actual movements for a task. For many tasks, movements are naturally expressed as desired positions and orientations of the hand (Benati et al, 1980). Also, robots tend to either (a) obtain movements and forces from a task description, or (b) obtain motor commands from desired movements and forces, but not both (Takase, 1979). That is, "low level intelligence" in arms has not been combined with the "high level intelligence" of problem-solving robots (p.1095). The skills of "comparatively special-purpose low intelligence systems" are "far more dexterous than those of intelligent robots" (p.1095). So the determination of motor commands from a task description may be treated in two parts: (a) the requirements for, and the state-of-the-art of the determination of movements and forces from a task description; and (b) the requirements for, and the state-of-the-art of the determination of motor commands from movements and forces. (a) is given in section 2.1, and (b) in section 2.2.

II.2.1 Description to motor commands

From a description of a movement task the required positions and orientations of the arm tip, and forces and torques exerted by the arm

tip, must be determined as functions of time (Benati et al, 1980; Albus, 1975 ; Dobrotin & Lewis, 1977).

The way existing robots determine movements, forces and torques from a task description is by the use of high level robot programming languages. The teacher gives a task description in the language. For example, Lozano-Perez (1983) describes a language with this statement in it,

PLACE BEARING1 SO (SHAFT FITS BEARING1.HOLE)

AND (BEARING1.BOTTOM AGAINST SHAFT.LIP) (p.837).

The language interpreter or compiler produces a specification of movements and forces, by applying the rules of the language to the given task description. In some systems a small part of the task description may be given by leading or guiding methods. For example, Holland et al (1979) describe how a teacher can show a robot the positions and shapes of objects by leading or guiding its arm tip over the objects. However, the task descriptions one can give to existing robots are not much removed from a specification of the required movements, forces and torques, as set out in the remainder of this section.

The sort of description of a task one would like to give a robot is a rough English description (Inoue, 1979). For example, given the rough English description "Spray-paint the car door blue", the required position and orientation of the tip of the arm must be obtained as functions of time. Continuous control of motion in space and time is required for spray-painting (McGhee, 1979). The spray nozzle will be at the tip or hand of the arm. Door geometry, optimal spraying distance and speed, and whether or not the door is moving along a production line must all be taken into account.

Given the rough English description "Screw the nut on the bolt", the torque used to screw the nut on must be determined from the bolt

diameter, metal characteristics, and the function of the nut and bolt assembly. The maximum tightening torque depends on this information.

There are many difficulties currently being experienced in making a system to determine the arm tip positions, orientations, torques and forces required for doing movement tasks. There is work on language understanding; problem-solving; knowledge representation, acquisition and use; high level manipulation programming languages; and vision (see Winston & Brown, 1979a; Raphael, 1976; Finkel et al, 1975; Takase et al, 1981; Dobrotin & Lewis, 1977; Ambler et al, 1975; Popplestone et al, 1980; Grossman & Taylor, 1978; Hasegawa & Inoue, 1979; Hasegawa, 1982). However, no existing robot can perform complex movement tasks in the real world, given the sorts of task description we are used to giving (Raphael, 1976; Takase, 1979; Benati et al, 1980; Nevins & Whitney, 1979; Lozano-Perez, 1979; Winston & Brown, 1979a; Hasegawa & Inoue, 1979).

Takase (1979) states that "... in general we are at the dawn of the study of the skill of robot arms." (p.1095). Morris (1982) states that "Today's robots are 'dumb'." (p.62). There is not a mature technology for creating intelligent robots to do jobs that are boring or dangerous to humans (Winston & Brown, 1979a). Any apparent understanding that an industrial robot has of its assembly line task is "mere fiction" (p.19 Paul, 1979). No program exists for turning a task-level description, one specified in terms of desired effects on objects, into a specification of the movements required (Lozano-Perez, 1983; McLaughlin, 1982). There is no functional programming of robots yet (Nevins and Whitney, 1979). One cannot say to a robot "This is what I want, you figure it out", and have the robot do so. Of task-level robot programming languages, McLaughlin (1982) states: "All such languages are still in the research phases, and show little hope of reaching full implementation in the near future." (p.9). In a comparative study of

robot programming languages, Bonner & Shin (1982) state that "A truly task-oriented language that conceals low-level aids like sensors and coordinate transformations from the user is an as yet unachieved dream." (p.87).

II.2.2 Movements and forces to motor commands

Having determined, or been told, the movements and forces that are required for a task, a robot must produce them. It must send motor commands to the control system or systems in its body which produce the desired movements and forces. That is, the robot must control its body so that the desired movements and forces are achieved. The determination of motor commands given desired movements is discussed in detail in the appendix. Briefly (i) kinematic equations describing the geometry of the arm can be solved to obtain joint movements, given desired hand movements; and (ii) dynamic equations may be solved to obtain the joint forces and torques that must be exerted, given the desired joint movements from (i). A PDP 11 computer can perform the solutions in real time as long as the characteristics of the load and arm are known. The determination of motor commands given desired torques and forces is also discussed in the appendix. Briefly, there are three aspects to controlling the forces and torques exerted by the arm tip: (i) determining joint forces and torques from the static equations of the arm, given tip forces and torques (Horn, 1979); (ii) controlling the compliancy of the arm (Simons, 1980; Mason, 1981); and (iii) sensing and servoing tip forces and torques (Simons, 1980; Mason, 1981).

The control systems of existing robots are not as good as those of humans. "Current robots must rely on control mechanisms slower than a human's and far less flexible." (p.v Dodd & Rossol, 1979).

II.3 EXISTING METHODS FOR TEACHING ROBOTS

Industrial robots are difficult to teach (Rosen, 1979; Grossman & Taylor, 1978; Nitzan, 1979). The teaching involved in setting up batch production runs and in making product model changes, is expensive. Expert teachers and programmers are often required for teaching industrial robots, rather than the teaching being done by unskilled people or people skilled in the tasks.

Robots are (i) programmed to do tasks, using programs which specify movements, forces, torques and motor commands, (ii) physically led through the sequence of movements required for a task, and (iii) guided through the movements via a teaching device. Section 2.1 explained that existing robots are not programmed with task descriptions.

The existing uses of the leading method are described in section 3.1. The guiding and programming methods are described in section 3.2. A robot may use a combination of methods, as explained in section 3.3. The facility of each method, and the range and difficulty of the tasks that can be taught with each method, are given (a) at the end of section 3.1, for leading, and (b) in the appendix, for guiding and programming methods.

II.3.1 Leading

The leading method is one way for a robot to acquire an internal representation of a manual task (Simons, 1980; Treer, 1979; Allan, 1979; Astrop, 1982 ; Benati et al, 1980; Haugan, 1974; Haugan & Jarvis, 1974; Vaccari, 1982; Grossman & Taylor, 1978; Holland et al, 1979; Raphael, 1976; Hohn, 1979; Lozano-Perez, 1983; Ambler et al, 1982).

The leading method is characterised by the teacher doing the task movements himself, with the robot's arm, rather than having any form of controller to operate. The robot remembers the movements. An example of a task which a robot can acquire by being led by the hand is the task

of spray-painting an object (Haugan & Jarvis, 1974; Haugan, 1974; Vaccari, 1982; Production Engineer, 1982; Compressed Air Magazine, 1981; The Industrial Robot 1982). While the robot is in a passive training mode the teacher paints an object by physically moving the robot's hand through a sequence of painting movements, and spray-gun trigger presses. The movements and trigger presses are recorded. They are played back to the arm at some later time; the robot paints the object. Allan (1979) states that "The most common method of teaching a robot is literally to lead it by the arm through the required sequence of operations." (p.32).

Beecher (1979) describes an assembly line system which has several arms. One arm is used for training and the rest for assembling. The training arm is identical to, and interchangeable with the other arms.[1] It is led or remotely guided through tasks. The sequences of motor commands the training arm has recorded during leading or guiding, may be replayed by the other arms.

Some tasks cannot be acquired solely by a robot being led through the task movements. This is because the teacher shows the robot only the movements required, but not the motor commands required for achieving the movements. As explained in chapter III, some led motor commands must be edited, so that certain opposing forces are counteracted. Leading does not show a robot how to counteract opposing forces. The teacher's leading solves only the kinematic equations of an arm (Benati et al, 1980). The teacher's leading does not determine the torques and forces that must be exerted by the arm's actuators.

There are various criticisms made of the leading method. The main one is that the industrial implementations of the leading method do not really make available the logical and symbolic commands that can be used extensively in programming languages (see Simons, 1980; Seltzer, 1979; Lozano-Perez, 1983; Ambler et al, 1982). Also, it is more

difficult to edit and document led tasks (Simons, 1980). For example if a movement must be added to, or deleted from a sequence of led stored movements, most existing robots require all the subsequent movements to be led again (Seltzer, 1979). In addition, some programming languages allow "... flexible use of sensor information for adaptive control" (Seltzer, 1979).

Programming facilities may be added to the leading method to overcome these disadvantages. For example, Takase et al (1981) describe a system that acquires tasks by a combination of lead through and high level programming. Tasks are programmed, leaving some coordinate transformations undefined in the program. The transformations are taught by the teacher leading the arm to the appropriate position, whenever the program hasn't enough information to prescribe a particular movement.

Led movements may be converted into MCVE statements in a high level manipulation programming language (Jarvis, 1982). Once a sequence has been led, the resulting program may be edited, and documented, to form a complex task.

In this thesis I describe improvements to the leading method which have quite a different nature. In chapter III I describe how the leading method can be enhanced so that a robot can select its own motor commands for producing a sequence of movements in a sequence of environmental conditions. In chapter III I also describe how the leading method can be enhanced so that a robot can select its own motor commands for achieving goals, rather than repeating a fixed, led sequence of motor commands or a fixed sequence of led movements. The robot can select motor commands on the basis of sensory information. A verbal correction scheme, itself set up by the leading method, can be used by a teacher to correct a robot's movements and motor commands during the performance of a task. There is no programming; all

movements and motor commands are taught using either leading or verbal correcting. Goals are set by a teacher when he leads or verbally guides a robot to them.

This work in chapter III overcomes another criticism made of popular leading: that it is sufficient only when the positions and movements are known at teaching time (Lozano-Perez, 1983). The robot can seek goals, instead of following a fixed sequence of movements.

On-line programming by guiding, to be discussed in section 3.2, and leading, have been criticised in a comparison with off-line programming methods, for a third and fourth reason:

> The first inefficiency is in the use of the human progammer. On-line programming of complex tasks is usually tedious, can be imprecise, and may be dangerous for the programmer... ... The second inefficiency is in the use of the robot. On-line programming of complex tasks is time consuming and prevents the robot from doing useful work during the teaching period. (p.492-3. Ambler et al, 1982).

Now firstly, programming languages can also be tedious and time-consuming to use. For example, an experienced paint-sprayer would find it very tedious and time-consuming to program a spray-painting robot, using a robot programming language. In comparison with using a programming language, he would be unlikely to consider it tedious to lead a spray-painting robot through spraying tasks. So the use of a human to lead or guide robot's on-line may not be inefficient. Secondly, a robot will often need to be used during the debugging and development of an off-line program for a task. Note that in both off-line and on-line programming, one robot may be used for the teaching, while others perform tasks (Beecher, 1979), as explained above in this section.

Leading has been discussed by MacDonald (1979; 1981; 1982a; 1982b). Leading is discussed in detail in chapters III, IV and V, in which various improvements to popular leading are explained and investigated.

The method of leading a robot through tasks has been employed in a simple, real, single-jointed "arm-robot" both without goal-seeking (MacDonald, 1981) and with a simple form of goal-seeking (MacDonald, 1982a). The arm-robot is explained in chapter V. Illustrative interactions with the arm-robot are reported there. The way leading might teach a robot has been studied with simulated robots, by Andreae (1980a; 1980b; 1982a), and with a simulated version of the arm-robot (MacDonald, 1981; 1982a).

II.3.1.1 Facility, range and difficulty of tasks taught by leading

I pointed out at the start of section 3.1 that leading is an easy teaching method to use. The teacher uses his own natural motor skills to show the robot movements for a task. However, the popular implementation of the leading method enables only specific sequences of movements to be shown. So conditional branches cannot be put in the sequences by leading. Any branches in the sequence must be explicitly programmed in by the teacher, in the way explained in chapter III. Thus, as the amount of branching required increases, the teaching becomes more and more a matter of programming, rather than one of leading. The teacher no longer uses only his natural motor skills. So only a fixed sequence of movements can easily be taught to a robot using the popular implementation of leading. In chapter III I explain how this can be overcome in two ways: (a) with a goal-seeking (GS) system, and (b) with a production system of corrections (PSC). No programming is required. The robot selects its own motor commands---taught by leading---for achieving goals, in the case of a GS system, or for performing a sequence of movements, in the case of a PSC.

The range of tasks that can be taught using the popular leading method is somewhat restricted. Any task that can be performed by the execution of a sequence of movements can in principle be taught using popular leading, as long as (a) the arm's control system automatically compensates for forces which oppose the movements, (b) the teacher can lead the movements, and (c) the robot is physically capable of performing the movements. I explain in chapter III a verbal correcting scheme (VC) which enables the first two limitations to be overcome in most tasks. Verbal corrections can be made so that errors in the recorded sequence can be corrected.

Tasks which require a robot to exert specific forces in the environment, regardless of the resulting movements, cannot be taught using the popular leading method. The teacher can show only specific movements, not specific forces. The VC method that I explain in chapter III might be used to teach a robot to perform led motor commands for exerting specific forces. The teacher could say "up" to make the robot push harder upwards, when the robot's arm sags down. This method of teaching a robot to use specific forces may be cumbersome. It is nevertheless possible.

II.3.2 Programming and Guiding methods

In using these methods a teacher interacts with the robot via a teaching device, for example a keyboard or joystick. There is no fundamental distinction between programming and guiding. They differ in the degree of real-time feedback that there is to the teacher or operator about what the robot is doing.

Near the extreme of minimum real-time feedback to the operator is the numerically controlled machine. A tape program may be prepared for a numerically controlled machine before the task is executed.

Near the extreme of maximum real-time feedback to the operator is a master/slave controller system (Sheridan et al, 1979; Vaccari, 1982;

Morris, 1982), where the forces exerted on the robot's body are relayed to the teacher's body and the teacher is provided with visual feedback of what the robot sees. The teacher remotely guides the slaved robot through a task. Both the sensory information from the environment of the robot, and the command signals, the motor commands, to the slaved robot are recorded. The robot executes these motor commands later, by itself. For example one can imagine an operator "showing" such a robot how to ride a bicycle. The operator is strapped into a master controller so that all his body movements can be measured and duplicated by the robot. The visual information that the robot gets is relayed to the operator's eyes. The forces exerted on the robot's body are "reflected" to the operator's body. The operator might actually feel as if he is riding the bicycle himself, which in a sense he is.

Other guiding and programming methods are described in the appendix. It can be difficult for us to translate our knowledge of skills into programs for robots (Michie, 1979); and there can be an explosion of declarations about the real three dimensional world when programming a robot with a high level manipulation language (Hasegawa and Inoue, 1979; Hasegawa, 1982).

Off-line programming does not provide the simultaneous "N-dimensional digitising capability" that leading and guiding provide (Ambler et al, 1982). Ambler et al state "This raises the frightening possibility of an unmanageable calculational load on the programmer." (p.492). They go on to discuss the RAPT system, which enables "the programmer to specify the task in an especially natural way". However, neither RAPT nor any other system enables a teacher to use a rough English description of a task to program a robot, as set out in section 2.1.

Nagel (1983) states that "Because robots operate on factory floors, some feel programming languages must be avoided. Although this is not

true, as indicated by the advent of the personal computer and the invasion of computers into many unrelated fields, the fear of programming them continues. Experience can eliminate such problems." (p.80). However, using a personal computer is not the same as programming robots. Robots act in the real, multidimensional world, in real time, interacting with other real world objects.

Many authors discuss, suggest, describe and implement improved robot programming languages (Taylor et al, 1982; Ambler et al, 1982; McLaughlin, 1982; Takahashi & Kohno, 1982; Park & Burnett, 1979; Finkel et al, 1975; Takase et al, 1981; Grossman & Taylor, 1978). Robot programming languages are surveyed by Bonner & Shin (1982) and Lozano-Perez (1983). In this thesis I take a different approach to robot teaching. Chapter III describes ways of improving the leading method that involve no use of robot programming languages by the teacher.

More details of guiding and programming methods are given in the appendix.

II.3.3 Combinations of methods

Combinations of guiding and programming, and the leading method, may be employed. For example, leading might be servo-assisted. Servo-assisted leading is a combination of the leading and guiding methods. The teacher provides some of the force required for making a movement. The servo-assistance provides the rest.

The method of leading a robot through movements has been used in several ways, in addition to the recording of a sequence of movements to be played back to the arm. Holland et al (1979) describe an arm system that is shown grasp points for parts by having its hand manually placed at the grasp point. The system picks up unoriented parts from a conveyor belt. A vision system determines the position and orientation of the parts. The system described by Takase et al (1981), mentioned in

section 3.1, has most of a task programmed. Some movements are led. They are incorporated into the program. Cunningham (1979) and Takahashi & Kohno (1982) both describe led robots which have programming features for editing led sequences, and for implementing program loops and conditional branches.

II.4 DETERMINING MOTOR COMMANDS FOR DIFFERENT CONDITIONS

This section examines the requirements for, and the state-of-the-art of performing a task in a variety of situations. There are three parts: (a) varying motor commands to cope with a range of environmental conditions (section 4.1); (b) uncertainty about the environmental conditions (section 4.2); and (c) the use of sensory information (section 4.3). Uncertain environmental conditions are those which a teacher does not, or cannot forsee.

II.4.1 Motor commands for different environmental conditions

A task will often need to be performed under a wide variety of specified environmental conditions. Some environmental variation can be eliminated. Even then, it can be more expensive to do this, than to cope with the environmental variation (Clocksin et al, 1982; Zecha et al, 1982). Also, environmental variation cannot be avoided in some tasks; for example in welding, and in chocolate decoration, where

delicate chocolates cannot be put through part feeders (Cronshaw, 1982).

Consider spray-painting tasks. One might expect a spray-painting task to be done with spray nozzles of different size, shape and weight and with different spraying characteristics. If these differences significantly affect the dynamic and static equations of the arm then they must be taken into account when solving these equations. A robot may use differently shaped tools. So the kinematic equations, in terms of the tip of the tool, may also change. For example, paint spraying nozzles may vary in length. Thus different situations may require different dynamic, static and kinematic equations to be solved. That is, coping with a range of conditions may involve achieving one particular sequence of movements despite changes in the conditions of the task.

Different situations may also require different movements, torques and forces to be achieved. For example, obstacles in the working environment of a spray-painting robot must be avoided by the robot. Different working environments may have different obstacles to be avoided at different locations. So different arm trajectories must be achieved in these different environments. Another example is the task of riding a small bicycle, having acquired the skill of riding a large bicycle. Aspects of the situation must be taken into account when obtaining arm tip positions, orientations, torques and forces from a specification of the task.

Present day industrial robots are severely limited in that they can carry out movement tasks only in a very limited range of situations (Benati et al, 1980). There are industrial robots that can easily be trained to do tasks under fixed conditions, but they are not adaptable (Nitzan, 1979). Advanced robots, on the other hand, lack basic movement skill but have the ability to cope with a wider range of situations (Raphael, 1976; Takase, 1979). We are a long way from having a robot

that can perform a wide variety of tasks in a wide range of situations. It is difficult to make a stand alone robot to cope with the four aspects of automobile spray-painting tasks that are listed below (Engelberger, 1979):

(i) observing the colour of the next car as it comes to the robot for painting;

(ii) selecting the appropriate paint;

(iii) holding doors, trunks, lids and hoods at convenient positions for painting;

and (iv) adapting to varying assembly line speeds while observing and applying proper paint.

Robots also require supporting aids such as feeder mechanisms for part presentation (p.v Dodd & Rossol, 1979).

II.4.2 Coping with uncertainty about the environment

The range of environmental conditions under which a robot must perform a task may not be completely specified. In general the environment in which a task is to be performed cannot be completely determined beforehand (Shirai, 1979a; Benati et al, 1980; Chien and Weissman, 1973; Whitney & Junkel, 1982). A robot that has to accommodate a wide variety of inputs with very little a priori information must be able to receive and respond to information about the changing environment (Dobrotin & Lewis, 1977). Dobrotin and Lewis describe an arm system for sampling soil during space exploration missions. It is very difficult for the designers to know what the environment will be like so the robot must be able to cope with a certain amount of uncertainty.

Holland et al (1979) describe a system for manipulating parts of unknown orientation. This system might be required to handle parts with shapes which the designers hadn't specifically designed the robot for.

As explained in section 4.1, existing robots are severely limited in their ability to cope even with a range of known environmental conditions. Existing industrial robots cannot react to "unforeseen circumstances" (p.35. Ayres & Miller, 1982).

Robots "have almost no error recovery" (p.v Dodd & Rossol, 1979). Control programs for industrial assembly robots must take into account the possibility of errors in object placement and shape (Brooks, 1982).

II.4.3 Using Sensory Information

In coping with both known and unknown environmental conditions a robot may need to use sensory information.

Industrial robots lack the ability to use sensory information for performing a manual task in a variety of situations. "To date there does not exist a commercially available, fully progammable industrial robot capable of using, as needed, all the available sensory feedback systems, and in particular the machine vision system." (p.5 Rosen, 1979). "State-of-the-art robots" have limited capability to coordinate their arms with sensory information (Ayres & Miller, 1982).

Visual information is required for doing some motor tasks; for example, hitting a tennis ball and picking up an object with a priori unknown orientation and position. There is a great deal of activity in the field of computer vision (Hanson & Riseman, 1978; Lerner, 1980), in particular on robot vision (Dodd & Rossol, 1979; Lerner, 1980; Kruger & Thompson, 1981). A vision system is seen as an essential part of a robot that is to perform a variety of complex movement tasks in a wide range of situations (Horn, 1979; Dodd & Rossol, 1979; Birk et al, 1981). However, making such a vision system has proved difficult (Rosen, 1979; Lerner, 1980; Shirai, 1979b; Reddy & Hon, 1979). The general visual control problem is still far from solution (McGhee, 1979). "... the study of computer vision is still in its infancy." (p.1524 Kruger & Thompson, 1981). Compared to human vision, machine

vision is rudimentary (Tenenbaum, et al, 1979; Horn, 1979). In this thesis I am not concerned with the difficult problem of providing vision for a robot, but only with enhancing the design of teachable adaptable robots.

Industrial robots lack sensors, so their capacity to respond to conditions in the environment is limited (Takase et al, 1981; Lee, 1982). The human performs so well because of the richness of his sensory feedback, rather than the mechanical efficiency of his arm, which is not high (Rabischong et al, 1977).

II.5 WAYS FOR EXISTING ROBOTS TO COPE WITH A RANGE OF SITUATIONS

As explained in section 4.1, coping with a range of situations may involve (i) achieving a sequence of movements despite changes in conditions, and (ii) achieving different movements, torques and forces under different environmental conditions. Ways to provide compensation for disturbances, so that a particular movement is achieved under a variety of enviromental conditions, are discussed in the appendix. The ways are: inherent and synthesized elasticity, viscosity and inertia in the arm; fixed parameter compensation; and adaptive control. The achieving of different movements, torques and forces is discussed in section 5.1.

II.5.1 Determining different sequences of movements

One way for a robot to be able to perform a task in widely varying situations is for it to be led, guided or programmed to do the task for each possible situation. However, some robots may be able to cope with changes in situation without having to be reprogrammed, reguided or reled through the task.

Imagine a robot with a model of the world and of the effects of its actions on the world. Imagine that the robot can update its model of the world when the world changes and that it can use the information in the model to work out a way of achieving some task. The task might be "ride the bicycle down the street". If the robot has a good model and can use the model effectively then one can imagine it coping with different sized bicycles, once it has been guided or programmed to ride bicycles of a particular size.

Dobrotin and Lewis (1977) describe a robot with built-in knowledge about itself and its environment. The robot does not deal with manipulation dynamics. This robot was mentioned in section 4.2. It has the ability to alter its knowledge on the basis of sensory information. The robot uses its knowledge of the environment and of itself to plan arm trajectories for doing soil sampling tasks. It is designed to cope with obstacles and other characteristics of the sampling site which can't be known in advance. The robot avoids objects in the environment, having obtained information about the obstacles from its sensors. So the robot acquires a task partly as a result of the information it obtains during its own activity. It can adapt to a change in environmental conditions without help from the teacher. The basic movements are programmed into the robot. However, arm trajectories are planned and executed on the basis of information accumulated while the robot is active in the environment. I will refer to this method of partly acquiring a task as the "active learning" method. The robot learns

while it performs movements. Leading is an example of "passive" learning.

A robot might acquire the task of riding a bicycle by active learning. The robot can be led through steering movements for steering the bicycle. It must obtain and respond to information about the environment in order to be able to use the steering movements to keep the bicycle balanced. In this case the robot acquires the steering movements by being led through them, but acquires the skill of riding the bicycle by obtaining and responding to information about the balance of the bicycle.

Chapter III explains how a led robot with a (GS) system in it can learn actively. For example, the information the demonstration arm-robot of chapter V obtains during active execution of movements, enables it to better perform the task.

Just as Dobrotin and Lewis's robot system does, the example robot systems mentioned at the end of the appendix are able to perform an extremely wide range of different movement sequences in different situations. However, they are severely limited in that they are specific to particular tasks and the environments associated with those tasks. They are specifically programmed to do a particular task. Extending such systems may not be straightforward (Waltz, 1982). Their important contribution to the design of robots is to show that it is possible to get a robot to do the particular tasks.

II.6 CONCLUSION

The teachability of a robot is concerned with the interaction between the robot and the teacher, in giving descriptions, telling and showing movements and forces, telling and showing motor commands, and helping a robot determine motor commands given descriptions and movements and forces.

Most teachers will agree that the greater the range of tasks that can be taught, the greater the difficulty of tasks that can be taught, and the easier the teaching, the greater the teachability. Existing robots lack teachability in all three ways. Existing robots are taught using leading, guiding and programming methods.

The adaptability of a robot is to do with how the robot copes with different environmental conditions. In determining the motor commands required for a task from a description or from movements and forces, a robot may take into account information about the environmental conditions.

Most humans will agree that the greater the range of conditions a task can be performed in, the less teacher assistance required when there is a change in conditions, and the quicker the robot's adaption, the greater the adaptability. Existing robots lack adaptability in all three ways.

NOTES FOR CHAPTER II

1. Some painting-robot manufacturers use a special light-weight training arm which a teacher leads through movements. The movements are played back to another, different arm, which does the actual painting. The makers of the Trallfa robot, which has 75% to 80% of the painting-robot market, have rejected the use of a special training arm (The Industrial Robot, 1982). They claim that training arms lack rigidity, hinder design modifications, have calibration problems and are expensive. So the actual robot arm is a better one to use for leading. The new Trallfa robot is specially designed so that the arm is easy to move and counter-balanced, during leading.

APPENDIX FOR CHAPTER II

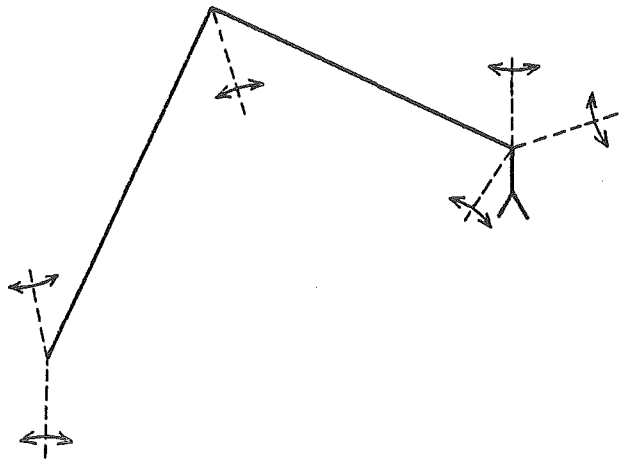
II.A.1 Movements to motor commands

Actuators apply torques and forces to the joints of an arm, rather than directly controlling the position and orientation of the tip or hand (Benati et al, 1980; Albus, 1975 ; Horn, 1979). So required joint torques and forces must be determined from given tip positions and orientations.

The kinematic equations of the arm may be solved for joint positions and orientations given tip positions and orientations (Paul, 1981). The dynamic equations of the arm and load may be solved for joint torques and forces given joint positions and orientations (Hollerbach, 1980).

The kinematic equations describe the basic geometry of the arm (Horn, 1979). An example six-jointed arm is shown in Figure II-8 (a). An example tip or hand coordinate system is shown in Figure II-8 (b).

(a) Example joint coordinates for a robot arm.



(b) Example hand coordinates (position and orientation/forces and torques)

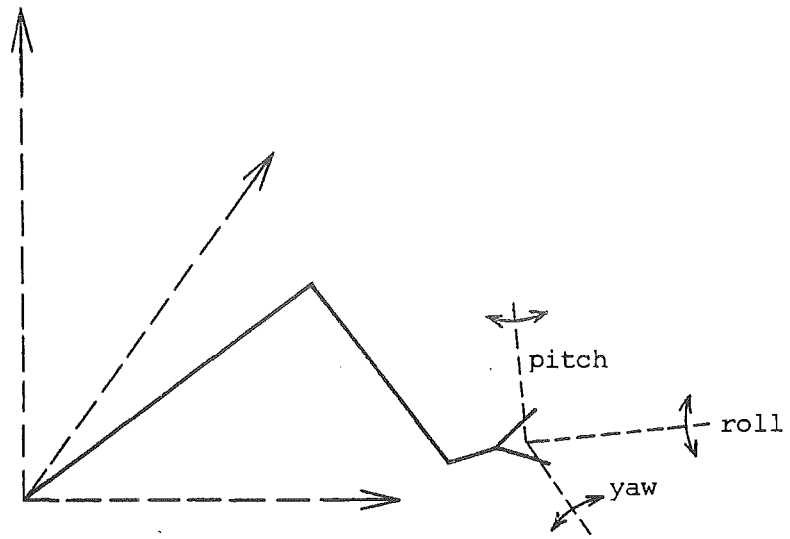


Figure II-8 Example coordinate system

The dynamic equations relate applied joint torques and forces to joint positions, velocities and accelerations. The dynamic equations take into account the effects of gravity, inertia, friction and the joint interaction forces which are centrifugal and Coriolis forces (Horn, 1979; Hollerbach, 1980; Raibert & Horn, 1978; Paul, 1981).

If the arm has more than six degrees of freedom then some constraints may be applied to arm movements. Six degrees of freedom are required for specifying the position and orientation of an arm tip in three dimensions. For example the redundancy in having more than six degrees of freedom may be required for doing tasks in environments where there are obstacles to be avoided, or where the working space is limited (Winston & Brown, 1979a).

Van Dijk (1978b) has suggested that spare degrees of freedom be used also to minimise (a) the energy expended during a movement, (b) side effects during a movement, and (c) the duration of a movement. Side effects are movement components in directions other than the desired direction, due to actuator torques applied about a joint. The side effects must be counteracted by torques applied to other joints so that the desired movement can be obtained. If there are spare degrees of freedom then there is some choice in selecting which joints to move. The joint that can move the most in the desired direction can be selected for producing the movement.

In general, solving the kinematic equations of a six degree of freedom arm for unknown joint positions is an intractable problem (Horn, 1979). The intractability is due to the presence of sines and cosines of joint angles in the polynomial terms of the kinematic equations. For certain arm geometries the solution is much easier; for example, if the last three rotational axes of a six degree of freedom arm intersect at a point (Horn, 1979; Hollerbach, 1982). The solution is tractable because the orientation and position of the tip can be

treated more independently. Luh et al (1980) present a control algorithm for a Stanford arm which has a cycle execution time of 11.5 ms on a PDP 11/45 computer. The algorithm combines the kinematic and dynamic solutions for the arm, whose last three rotational axes intersect at a point.

The problem of solving the kinematic equations of an arm may be avoided if the robot is guided or led through movements. The leading and guiding methods are discussed in section 3. The teacher moves the arm, either physically (the leading method) or by remote control (the guiding method), so that the joint movements may be recorded. However, it may be necessary to record the movements in hand coordinates, as does the verbal correcting system discussed in section 2 of chapter III. In that case, the hand movements must be converted back into joint coordinates for the movements to be performed.

The dynamic equations of an arm with six degrees of freedom can be solved in real time (Hollerbach, 1980). A PDP 11/45 computer can perform the real time solution, as mentioned above. The coefficients of the dynamic equations depend on the configuration of the arm and the velocity of the joints (see Hollerbach, 1980; Paul, 1981; Horn, 1979; Albus, 1975 ; Raibert, 1978). So the instantaneous position and velocity of the arm must be known or measured. The dynamic equations also depend on the mechanical properties of the arm and its load. So the characteristics of the load on the arm must be known or measured in order to solve the dynamic equations. Ways for estimating the parameters of an arm and its load are discussed in section A.4.4.

Wu and Paul (1982) have discussed a method for controlling a robot arm which does not explicitly solve the dynamic equations of the arm. Instead the hand trajectory is specified, and then hand forces for achieving that trajectory are computed. The mass distribution of the load in the hand must be known in order to calculate the hand forces

required. Once the hand forces are known, joint forces are calculated from a static model of the arm. A static model relates static joint forces and torques to static tip forces and torques (Horn, 1979). The dynamic characteristics of the arm are ignored at this stage. During execution an on-line stochastic approximation method is used to adjust the applied joint forces so that the required hand forces are achieved. The actual hand forces are sensed at the wrist. The more the load dominates the dynamic characteristics of the arm itself, the better the control scheme works. The known load dynamics are taken into account in the calculation of required forces, so the more the load dominates the arm's unknown dynamics the better the control achieved.

Vukobratovic and Stokic (1982) describe a computer-aided method for an operator to determine the motor commands required for movements. The dynamic equations of the arm and load, global stability and local stability are taken into account. The operator may choose from various methods which the computer may use to deal with dynamics and stability. The synthesis of the parts of the controller is discussed in more detail in Vukobratovic et al (1981). The saturation of actuators is taken into account.

Freund (1982) discusses a method for determining a control law for each joint, taking into account the arm's dynamics, and the desired position and velocity feedback gains of each joint.

It is not necessary to bother solving the dynamic equations of arms that move slowly. Position and velocity feedback closed around each joint is sufficient to control a slow arm (Benati et al, 1980; Raibert & Horn, 1978; Hollerbach, 1980). However, fast moving arms are desirable in industry for the economy they bring from speed of operation (Raibert & Horn, 1978). Some tasks require the dynamic properties of loads on the arm to be used. For example, in order to chop wood an axe must be moved rapidly enough for its inertia to have a

considerable effect. If an axe were moved so slowly that its inertia had little effect on the movement then it would be no better for cutting wood than a pocket-knife, neglecting the effect of gravity.

Young (1978) applies the theory of variable structure systems to the design of an arm controller. The result is a nonlinear controller that can control an arm despite interactions between joints, and load variations, but without having to model or solve the dynamics of the arm. The control system needs to know only bounds on arm and load parameters, in order to ensure stability. Each joint is put into a "sliding mode" and kept there by a switching controller. Consider a one joint arm. The controller switches the arm control signal whenever the arm's behaviour crosses the switching line shown in Figure II-9. This causes the system to "slide" down the switching line, as shown in the Figure. In sliding mode, the behaviour of the arm is determined by the switching line, and is insensitive to joint interactions and load variations. The control signal depends on both the feedback signals from the arm, and which side of the switching line the system is on. Young discusses the design of such a controller for a multi-jointed arm.

The guiding methods, discussed in section 3.2, may avoid the need for dynamic equations to be solved. The teacher may remotely control the arm in real time. The joint actuator signals may be directly recorded and repeated later by the robot itself.

II.A.2 Forces to motor commands

Some tasks require forces and torques to be exerted on objects in the environment in a particular way. For example, object insertion tasks, and some other assembly tasks, require the arm to comply with the external forces in some directions, but not those in others (Nevins and Whitney, 1979; Inoue, 1979; Mason, 1981). The task of inserting a peg into a chamfered hole can be facilitated by having the arm comply

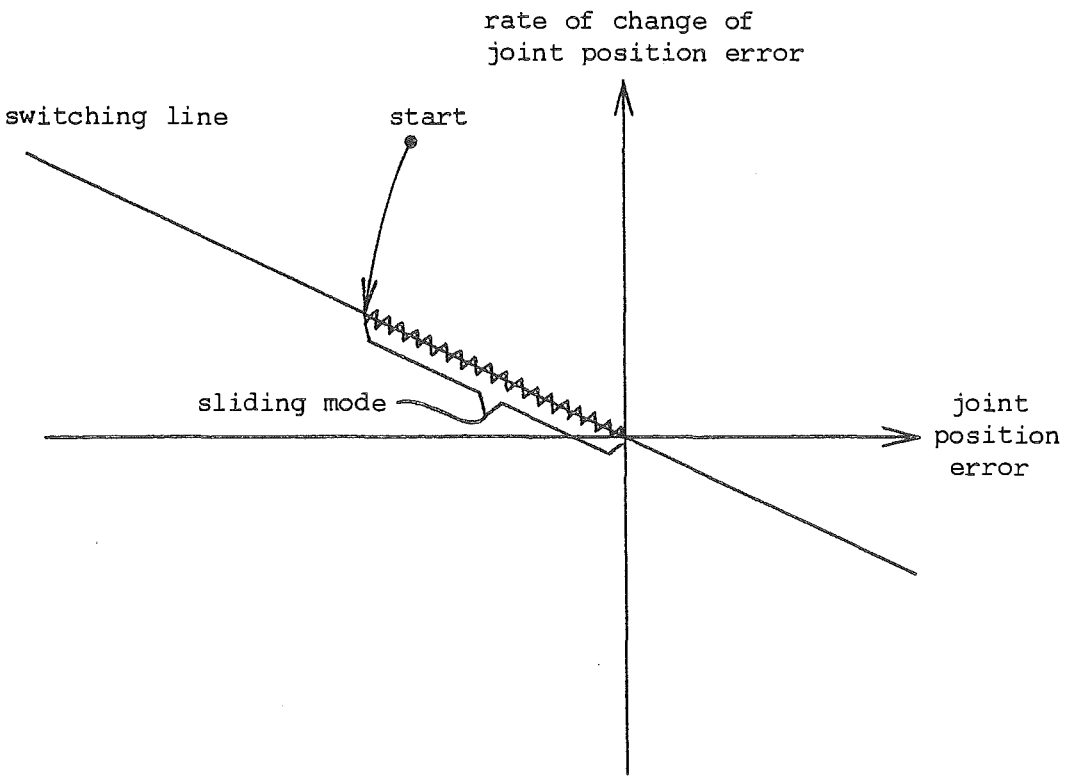


Figure II-9 Representation of "sliding mode" for a single jointed arm (see Young, 1978). The control signal to the arm is switched when the arm crosses the switching line, causing the system to "slide" down the switching line. Young developes a controller for a multi-jointed arm.

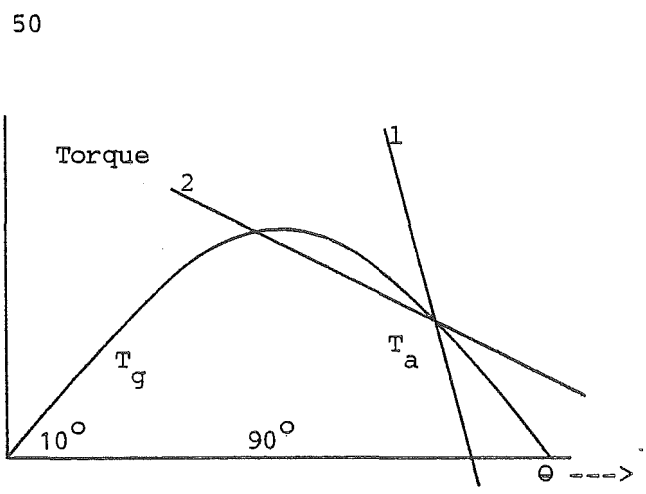
with forces in the plane perpendicular to the axis of the hole (Benati et al, 1980; Takase, 1979).

There are three aspects to controlling the forces and torques exerted by the arm tip: (i) determining joint forces and torques from given tip forces and torques (Horn, 1979); (ii) controlling the compliancy of the arm (Simons, 1980; Mason, 1981); and (iii) sensing and servoing tip forces and torques (Simons, 1980; Mason, 1981).

The static equations of an arm describe the relationship between the torques and forces exerted by the tip and the torques and forces at each joint (Horn, 1979). Solving the static equations of an arm for joint torques and forces, given desired tip torques and forces, is straightforward (Horn, 1979). These joint forces and torques can then be produced by the joint actuators. This method of achieving desired tip forces and torques is a feedforward method. The static equations are a static model of the arm. Compensation for gravity must be made. Friction and stiction are not taken into account by the static equations (Horn, 1979).

The compliancy of an arm is to do with the relationship between the arm configuration and the external forces and torques applied to it (Benati et al, 1980). The relationship can be in terms of joint forces and torques or in terms of arm tip forces and torques. The compliancy of an arm can be altered by altering the viscous, elastic and inertial properties of the arm. For example, consider the elasticity, or stiffness of the arm. Forces and torques exerted by the tip of the arm can be controlled by moving the arm against an object and trying to position it a certain distance inside the object. Contact forces and torques are produced. They depend on the elasticity of the arm and the distance inside the object that the arm is trying to move. The arm acts as a kind of spring (Horn, 1979; Benati et al, 1980). The inertial and viscous properties of an arm also cause resistance to external forces. Viscosity is a frictional property. The inertia of an arm determines the way the arm accelerates when torques and forces are exerted on it.

There are two ways to control the compliancy of an arm: (a) the gains of position, velocity and acceleration feedback loops may be controlled during the execution of a task, thus altering the compliancy introduced by feedback signals (Benati et al, 1980); and (b) the mechanical properties of the arm can be controlled at the design stage



$$T_a \propto -K\Theta$$

$$T_g = gL(M_a/2 + M_l)\sin\Theta$$

1 $K > gL(M_a/2 + M_l)$

2 $K < gL(M_a/2 + M_l)$

Figure II-10 A single degree of freedom arm is unstable under the influence of gravity; like an inverted pendulum (Benati et al, 1980). The terms used in the Figure are given below. Section A.4.2 of the appendix of chapter V explains in detail the equations and terms used in the Figure.

T_a	arm torque produced by arm actuator	T_g	torque produced by gravity
M_a	arm mass	M_l	load mass
g	gravitational acceleration	Θ	arm angle
L	length of arm	K	stiffness of arm

in such a way that the arm compliancy is suitable for tasks which would otherwise require compliancy control during the task (Nevins & Whitney, 1979; Mason, 1981). A mechanical device called a Remote Compliance Centre (RCC) can do mechanically what active force feedback does with sensors and servos.

An arm must have natural stiffness, or elasticity, so that it can be stable under the influence of gravity (Benati et al, 1980). As shown in Figure II-10, if the arm stiffness is not great enough then the arm will be unstable under the influence of gravity, as an inverted pendulum is. If the arm stiffness is great enough then when the arm is subject to small perturbations about an equilibrium point the change in

torque due to gravity will not be as great as the restoring torque produced by the stiff nature of the arm. Thus the equilibrium will be regained; it is a stable equilibrium point. The changes in gravitional torques produced by perturbations of the arm configuration will be greater for greater loads on the arm. The greater the load, the greater the stiffness required for stability.

The tip forces and torques can be controlled also by sensing the actual tip torques and forces, comparing them with desired tip forces and torques, and making appropriate adjustments (Inoue, 1979; Mason, 1981). The tip torques and forces are sensed and servoed. This feedback method of controlling tip forces and torques has two disadvantages (Benati et al, 1980): the force sensor becomes "the Achilles heel" of the system; and the force feedback control loop requires a large amount of computation, which does not seem warranted for the simple nature of some manipulation tasks.

II.A.3 Guiding and Programming methods

There is quite a range of programming and guiding methods that provide different amounts of real-time feedback to the operator about what the robot is doing. Off-line programming, say in an assembler language, provides very little feedback. High level manipulation languages (Takase et al, 1981; Finkel et al, 1975; Seltzer, 1979) may provide a more interactive way of programming a robot. For example, Park and Burnett (1979) describe an interactive compiler which enables a program to be resumed after being stopped and changed. Salmon & d'Auria (1979) describe an assembly robot that has its instructions programmed into it and its position data given using a joystick.

As mentioned in section 3.2, it can be difficult for us to translate our knowledge into programs, and declarations about the real world may explode.

Holland et al (1979) describe a manipulation system that is programmed to pick up objects from a conveyor belt, having "seen" them on the belt with its "eye". Object grasp points are shown to the system by its hand being manually placed at the grasp points.

Guiding methods can provide more real-time feedback to the teacher than the programming methods. Examples of guiding methods are: (a) guiding a robot through a task with a joystick (Sheridan et al, 1979; Beecher, 1979; Goksel et al, 1976; Seltzer, 1979; Freedy et al, 1971; Simons, 1980), (b) a keyboard (Sheridan et al, 1979; Morris, 1982; Nishimoto et al, 1983), (c) switches or buttons (Sheridan et al, 1979; Beecher, 1979), (d) a master/slave controller, as discussed in section 3.2, or (e) spoken commands (Simons, 1980; Nitzan, 1979).

The assembly line system described by Beecher (1979), mentioned in section 3.1, can have its training arm guided through tasks, as well as led.

Freedy et al (1971) describe a remotely guided robot arm controller which has a learning system in it. The operator guides the arm with a joystick. The learning system gradually takes over from the operator. It builds up a model of the task in the form of a set of conditional probabilities for arm movements from various arm positions.

Simulated robots with multiple context learning systems (MCLSs) have been preprogrammed to do movements. For example reflex eye movements can be automatically triggered by visual stimuli (MacDonald, 1979; Andreae, 1980a; 1980b; 1982a). Speech sounds can also be preprogrammed to happen in a reflex fashion, called "mimic-speech" by Andreae (1977a), and back-reflex by MacDonald (1979). Reflexes are discussed in chapter VII. Preprogrammed reflexes can result in the general learning of the reflex movements. Reflexes are not task specific. Basic movements are preprogrammed, not tasks. Section 3 of chapter VII discusses why reflexes are not as suitable for arm movements as they

are for eye movements and speech sounds.

The guiding method has been used for MCLSs in both simulated and real robots (Andreae, 1972-1983; Andreae, 1977a). This method is used for teaching a robot with an MCLS movements which it hasn't had preprogrammed into it.

The facility of guiding and programming methods, the range of teachable tasks, and the difficulty of teachable tasks, will now be discussed.

Most guiding methods are somewhat more difficult to use than the leading method, because it is generally more difficult to control a six degree of freedom robot arm via a keyboard or joysticks, than it is to physically lead it through movements. The master/slave controller does not suffer in this way. It may be expensive though.

The range and difficulty of tasks the guiding and programing methods can be used for are very broad. The guiding and programming methods can be used to teach a robot to exert a specific sequence of forces; something which the popular leading method cannot be used for, as explained in section 3.1.1. For example, using the guiding method a teacher may remotely control the robot so that forces are exerted. The motor commands used by the teacher to do this are recorded, and later played back to the arm.

As with the popular implementation of the leading method, the more complexity there is in the branching of the sequences for a task, the more the teaching is a matter of programming, rather than of on-line control.

Programming and guiding methods may be safer for teaching industrial robots than the leading method. The teacher may be neither physically in contact with the robot, nor near the robot.

II.A.4 Compensating for disturbances

Compensation for disturbances may be provided in four ways:

(a) by the inherent inertial, viscous and elastic properties of arm links and actuators (Benati et al, 1980; Nevins & Whitney, 1979) (section A.4.1);

(b) by the elasticity, viscosity and inertia provided by arm position, velocity and acceleration feedback signals, respectively (Benati et al, 1980) (section A.4.2);

(c) by fixed parameter compensation techniques such as introducing proportional, integral and derivative control into the arm control system (Elgerd, 1967; Langill, 1965; Kuo, 1972), and increasing the order of the arm control system (DiStefano et al, 1967) (section A.4.3);

(d) by an adaptive arm control system (section A.4.4).

For each of these methods section A.4.5 discusses the range of conditions coped with, the teacher assistance required and the speed of adaption.

II.A.4.1 Inherent elasticity, viscosity and inertia Section A.2 explained that an arm's inherent elasticity, viscosity and inertia determine the way external forces affect the arm.

Two examples of disturbances are a variation in the characteristics of an actuator and a variation in the load on the arm. The sensitivity of the arm to external disturbances is reduced by building elasticity, viscosity and inertia into it. However it may be undesirable for a robot arm to have so much friction and inertia that it cannot be controlled precisely and rapidly. The greater the inertia and friction inherent in the arm, the more power is required to move the arm. The frictional and elastic properties of the arm may be controlled more

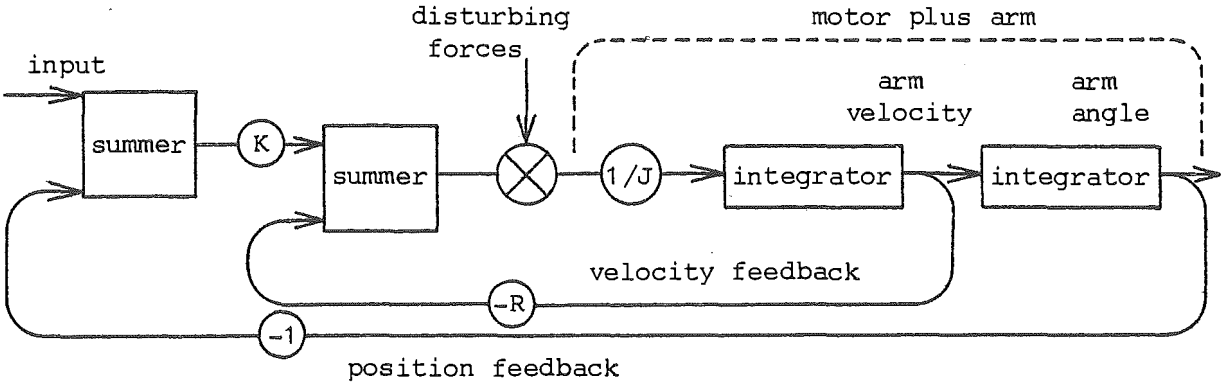


Figure II-11 A second order arm control system. J is the moment of inertia of the arm. Position and velocity feedback oppose disturbances.

easily if they are introduced by feedback signals rather than inherent in the arm itself.

II.A.4.2 Feedback The gains of position, velocity and acceleration feedback loops can be set to optimize the arm performance for a relatively narrow range of conditions. The feedback signals oppose disturbances. The optimal gains vary with the configuration of the arm and the load on the arm. Figure II-11 shows a single degree of freedom arm with position and velocity feedback. For this simple arm the optimal values of the position error gain, K, and the velocity feedback gain, R, depend only on the inertia of the arm and its load, as discussed in detail in section A.4.2 of the appendix of chapter V. Particular values of K and R are good only for small changes in the load.

II.A.4.3 Fixed compensation The characteristics of a second order system such as the one in Figure II-11 can be improved by adding integral error control and derivative control to the proportional control provided by the position error gain K (Elgerd, 1967; Langill, 1965). The effective value of K then becomes frequency dependent, for

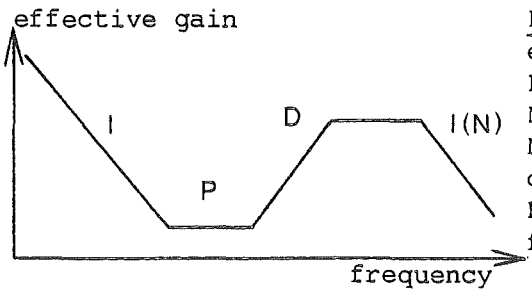


Figure II-12 Plot of effective position error gain, K, for a PID(N) controller:
Low frequency integral control, I
Mid frequency proportional control, P
Medium-high frequency derivative control, D
High frequency integral control, I(N) for noise reduction

example in the way shown in Figure II-12. A controller like the one plotted in Figure II-12 is a "lag-lead controller" with a high-frequency-noise filter (see Langill, 1965).

Integral control at low frequencies has the effect of eliminating any static error in the output position. The integral controller provides a compensating signal that increases more and more the longer an error persists. It may thus overcome static errors which the system in Figure II-11 cannot overcome (Elgerd, 1967; Langill, 1965). The integral controller eliminates the characteristic position error that the second order system exhibits when its input is changed at a constant rate. The output of a second order system with a linearly increasing input signal lags behind the input by a constant amount (DiStefano et al, 1967). An integral controller eliminates this lag.

Integral control at high frequencies helps to eliminate noise because it causes a reduction in the gain of the system for higher frequency signals. This decrease in gain at high frequency is shown in the I(N) section of Figure II-12.

Derivative control at medium-high frequencies improves the speed of response of the system (Kuo, 1962; Langill, 1965).

It is also possible to improve the performance of a second order system by introducing a third order into the controller (DiStefano et al, 1967). It is necessary for such a system to have acceleration feedback in order to be stable (Pipes and Harvill, 1970). A third order

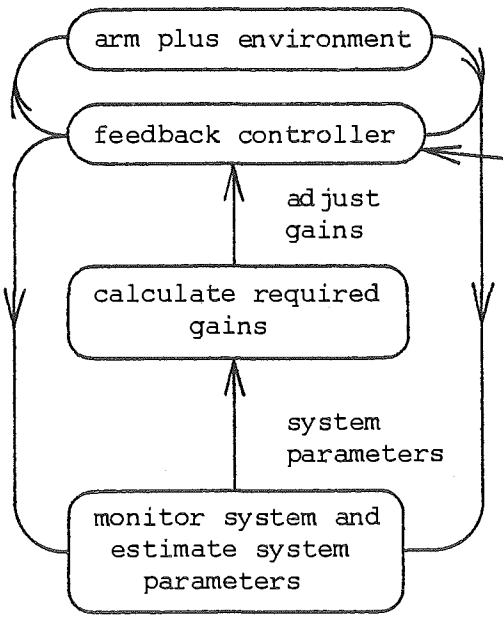


Figure II-13 An adaptive arm control system. (see Astrom et al, 1977). The feedback controller could be one of the systems shown in Figures II-11 or II-12. The gains of the feedback controller are adjusted to suit the arm plus environment characteristics. The charactistics change when, for example, the load on the arm changes.

system is similar to a second order system with low frequency integral control. A third integrating component is introduced in each case.

II.A.4.4 Adaptive compensation Controllers with fixed gain feedback can be improved by allowing the gain parameters to be adjusted. The fixed gain controller becomes a variable gain controller. Some way for estimating the required gains must be provided (Astrom et al, 1977; Mishkin and Braun, 1961). Figure II-13 shows the parts of such an "adaptive" controller. The performance of the system is monitored and its parameters estimated. For example the load on the arm might be estimated. Adjustments are made to the gains of the controller as the load, and hence optimum gains, change.

It is possible to have an adaptive system which has no feedback to the "feedback controller" box in Figure II-13. The commands are operated on by functions whose parameters are controlled by the "monitor system and estimate system parameters" box and the "calculate required gains" box.

Several authors have discussed adaptive compensation for an arm and its load (Raibert, 1978; Kirchman et al, 1977; Arbib, 1979;

Benati et al, 1980; Dubowsky & DesForges, 1979; Takegaki & Arimoto, 1981). Raibert (1978) and Kirchman et al (1977) have implemented schemes for updating tabulated coefficients of the dynamic equations. Updating occurs while movements are being performed. In Raibert's system a couple of thousand trials of a movement are required before the system "learns" to do the movement with a new load. Coefficients are estimated from measurements of the acceleration changes which occur when actuator signals change. Changes in actuator torques are compared with resulting changes in acceleration. Since the coefficients are indexed only for instantaneous joint positions and velocities, the system has to "learn" about new loads even if they have been manipulated before. No sensory information, other than arm configuration and joint velocities, is used to index the coefficient table. Coefficients for one load are overwritten when another, different load is manipulated.

Dubowsky and DesForges (1979) discuss a model reference adaptive control system that copes with changes in loads on an arm, keeping its dynamic characteristics reasonably invariant. The joints of the arm are controlled by second order controllers. The gains of the controllers are modified by the adaptive system. The reference model of the arm is a simple second order one. The adaptive controller has no a priori knowledge of the load on the arm. Takegaki & Arimoto (1981) also discuss an adaptive arm controller that compensates for a grasped load.

Albus (1979; 1975) describes a system which can be trained to control an arm. During a training stage the teacher provides (i) signals for the arm actuators, and (ii) command signals which he requires the system to learn to execute. The system adjusts a table of weights so that it can control the arm given just the command signals. Once trained, the system provides the arm actuator signals itself. Only a relatively small amount of training is required for the system to

learn to cope with a range of arm conditions.

Nishimoto et al (1983) describe an arm controller, with velocity and position feedback, similar to the one shown in Figure II-11. Unlike the controller of Figure II-11, its feedback gains are adjusted in order to minimize both the integrated error and the energy expended.

It has been suggested that test signals be applied to arms in order to estimate their parameters (Arbib, 1979; Benati et al, 1980). The mass and moment of inertia of an object are seldom otherwise available in the real world (Arbib, 1979). In the system used by Raibert the test signals were the actual movement signals. Specially designed test signals can be used to estimate various parameters more easily (Benati et al, 1980). Orthogonal sets of test signals enable independent measurements to be made. Small test oscillations across equilibrium configurations enable non-linear gravity terms to be linearized, facilitating the estimation of parameters from the test measurements.

II.A.4.5 Range of conditions, teacher assistance and adaption speed

All except the adaptive systems rely on fixed methods of coping with a range of conditions. Their performance does not change from one task performance to the next. For example, stiffness in an arm control system opposes disturbances. The greater the disturbance to the equilibrium position of an arm, the greater the opposition from the arm. However, the same disturbaance is always resisted in the same way.

It is not possible to make a formal distinction between an adaptive system and a fixed compensation system. This is because the adaptive system can always be expressed as a system with certain states, inputs and outputs, and a relationship between the states, outputs and inputs. It can be represented as a fixed system. However, it is useful to informally distinguish between arm control systems that change over a number of performances of a task, that is "adaptive systems", and those that don't.

Of the systems I have mentioned, only Albus's (1975; 1979) requires teacher assistance, as pointed out in section A.4.4.

The adaption speeds of the adaptive systems vary from a couple of thousands of trials of a movement (Raibert's (1978) system) to a few seconds (the systems of Dubowsky and DesForges (1979) and Takegaki & Arimoto (1981)).

Except for Raibert's and Albus's systems, all the systems cope only with those situations in which the arm's characteristics are still described by the same form of dynamic and kinematic equations. They cope when only the parameters of these equations change. However, the form of the equations may not stay the same. For example, the characteristics of a robot arm that steered a bicycle would depend on the bicycle, its speed, the road surface, et cetera. Raibert's system can cope with arm characteristics that can be modelled by a table of coefficients which are indexed by arm positions and velocities. The coefficients are for an equation of the form

$$T_m = J_a \times \ddot{\Theta} + K_{ab}$$

where J_a and K_{ab} are the coefficients, $\ddot{\Theta}$ is the desired acceleration of one of the arm's joints, T_m is the actuator torque that will be applied at that joint, a is the index into the table for the arm position, and b is the index for the arm's velocity. Albus's system also has a table of coefficients, or weights. The coefficients can be indexed by arm positions and velocities, and other sensory information.

II.A.5 Some examples of robot systems

A ten part water pump has been assembled by a system that uses force and touch, tools, and some simple vision (Winston and Brown, 1979a).

A robot system has been demonstrated that employs vision to enable randomly oriented cylinders to be acquired (Kelley et al, 1982).

An arm system which puts hoses onto water pumps has been installed in a factory (Uno et al, 1979). The hoses are used to pressure test the pump after its manufacture. The positions of the hose fittings on the pumps are only roughly known by the arm system. A vision system enables the arm system to put the hoses on by giving a better estimate of the position of the fittings.

Two co-operating arms have assembled a hinge (Winston and Brown, 1979a).

A bearing with 20 micrometre tolerances has been assembled by an arm whose positional accuracy is an order of magnitude worse than that tolerance (Inoue, 1979).

A robot system can transfer randomly oriented parts from a moving conveyor to a predetermined location (Holland et al, 1979). This system was mentioned in section 3.3.

Steel castings scattered on a light table have been identified and reoriented (Winston and Brown, 1979a).

An arm with touch and force sensors can feel its way as it packs assembled pumps into a case (Nitzan, 1979).

An arm system has built a structure made of various-shaped blocks, given an orthographic projection of the structure and some blocks on a table (Ejiri et al, 1972). One camera looks at the projection and another at the blocks and developing structure.

Another system built a block structure, having been shown an example of the completed version (Winston, 1972).

A robot system has assembled a toy car and a toy boat, given a jumble of parts for the two toys (Ambler et al, 1975).

A space exploration robot for doing general sampling handling tasks, assembly tasks and surface roving has been designed (Dobrotin and Lewis, 1977; Miller, 1977). This system was mentioned in sections 4.2

and 5.1.

An arm has assembled alternators using several tools and coping with some uncertainty in the location of parts in part feeders (Nevins and Whitney, 1979). The arm system takes less than three minutes to assemble an alternator. It has a compliant wrist and is either programmed to do movements or remotely guided through them via a "control box".

CHAPTER III

ENHANCING THE LEADING METHOD

My aim is to advance the design of teachable, adaptable robots. I will discuss the widely used "leading" (Simons, 1980; Compressed Air Magazine, 1981) or "lead-through" (The Industrial Robot, 1982)[1] method, of teaching robots manual tasks.

In this chapter two paths of improvement are proposed for the popular leading method of teaching robots. These paths are depicted in Figure III-1. The popular leading method was explained in section 3.1 of chapter II. A previously recorded sequence of commands is sent to an arm control system (ACS), as shown in Figure III-2. I shall call these commands both "motor commands", since they command movements, and "actions". For example, the teacher might lead the movement M, causing the motor command C to be recorded, as depicted in Figure III-2(a). The motor command C might produce the movement M, when sent to the ACS, as depicted in Figure III-2(b).

Both proposed paths of improvement deal with two specific limitations of the popular leading method. Firstly, in the popular implementation of leading, to change the motor commands stored during the leading a teacher must either re-lead the task or edit the motor commands via a keyboard or similar controller (Cunningham, 1979[2]; Jarvis, 1982). Editing the motor commands is awkward (Nagel, 1983). The more of this off-line editing a teacher must do, the less he is able to use his natural skill at doing the task. Also, the editing is not done as the robot performs the task.

Now, the need to change motor commands cannot be avoided. The

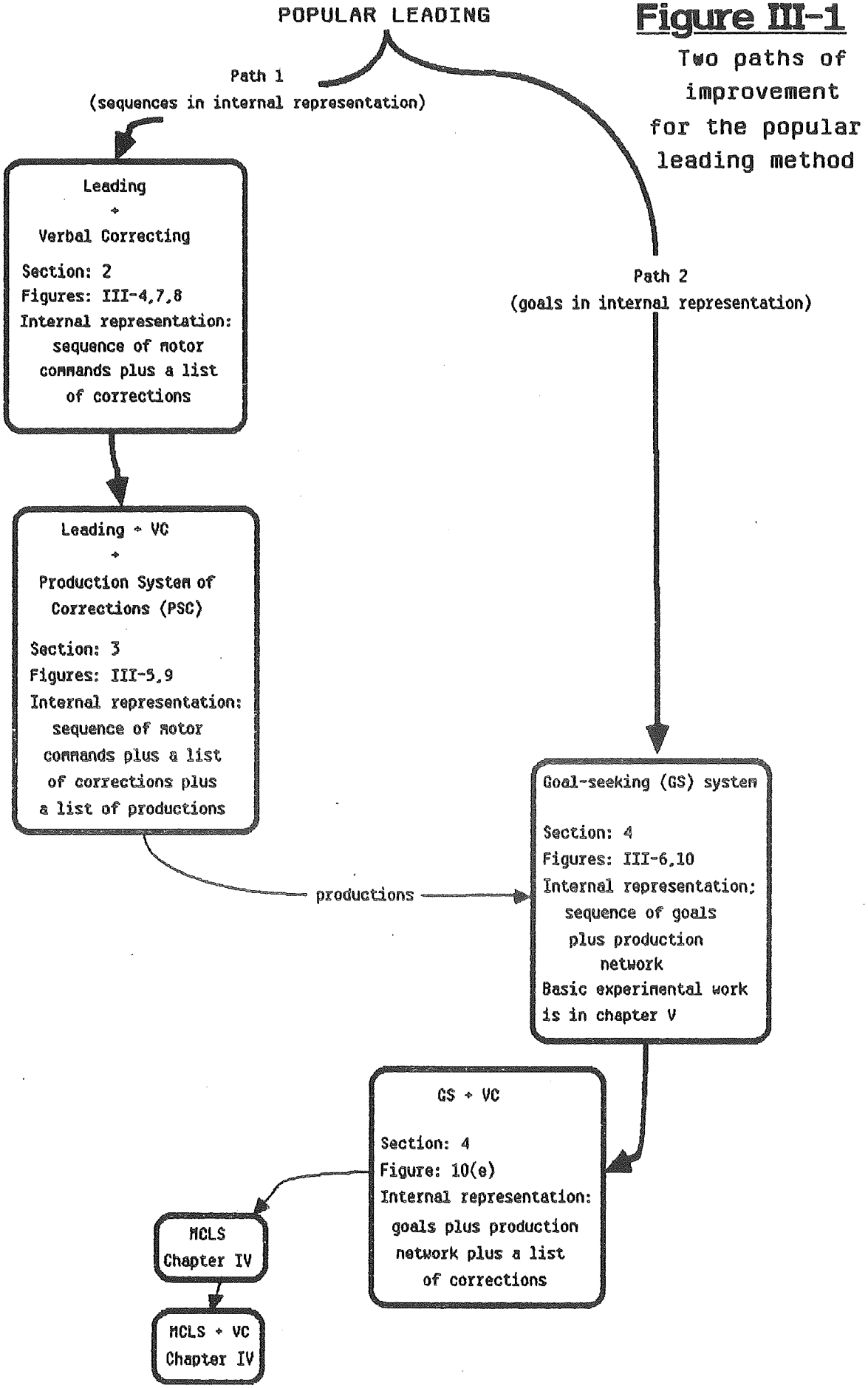
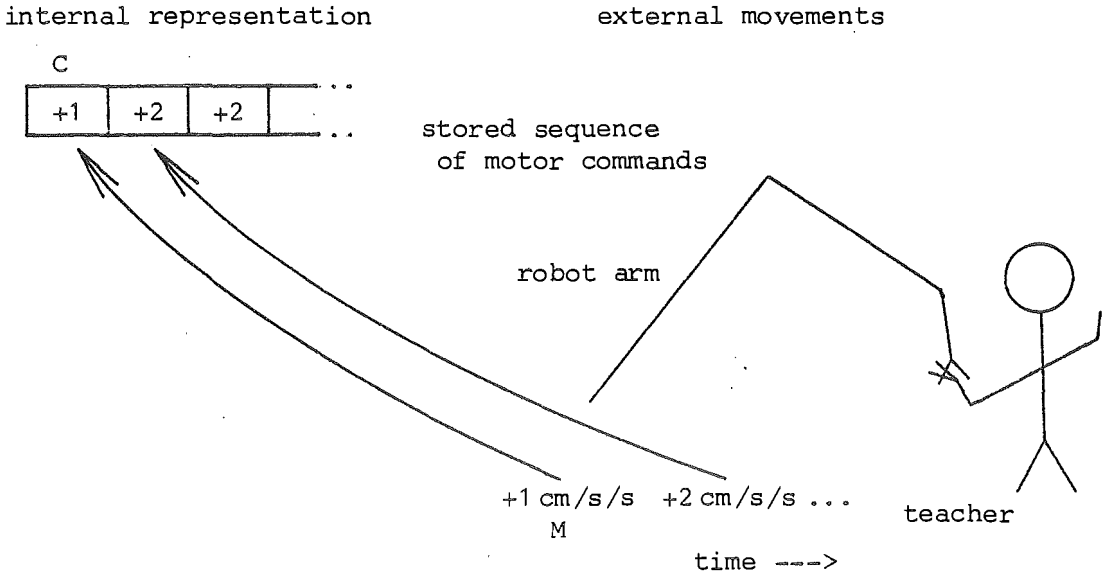


Figure III-1

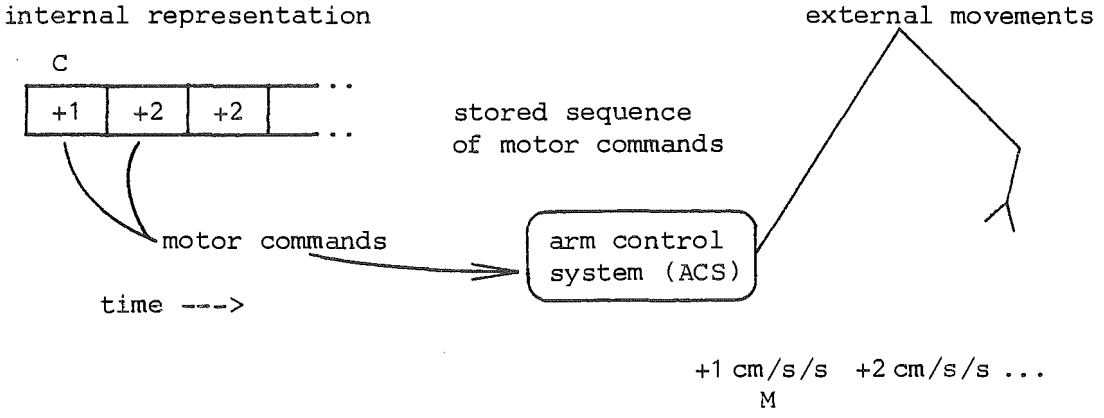
Two paths of improvement for the popular leading method

Figure III-2 Popular leading The movements and motor commands for one dimension are shown. While existing industrial robots may record movements in terms of positional changes, an internal representation of acceleration changes: (a) may be more suitable for future industrial robots with dynamics compensation, since the desired accelerations may be required (see for example Figure III-11 in section 2); and (b) may be more suitable for Verbal Correcting (VC), because accelerations are more closely related to opposing forces than positions are.

(a) Leading. A teacher physically leads the robot through movements, causing a sequence of motor commands to be stored in an internal representation. The internal representation is an ordered list of motor commands. As shown in the Figure, when a movement M, e.g. +1 cm/s/s, is led, a motor command C, e.g. +1, is stored.



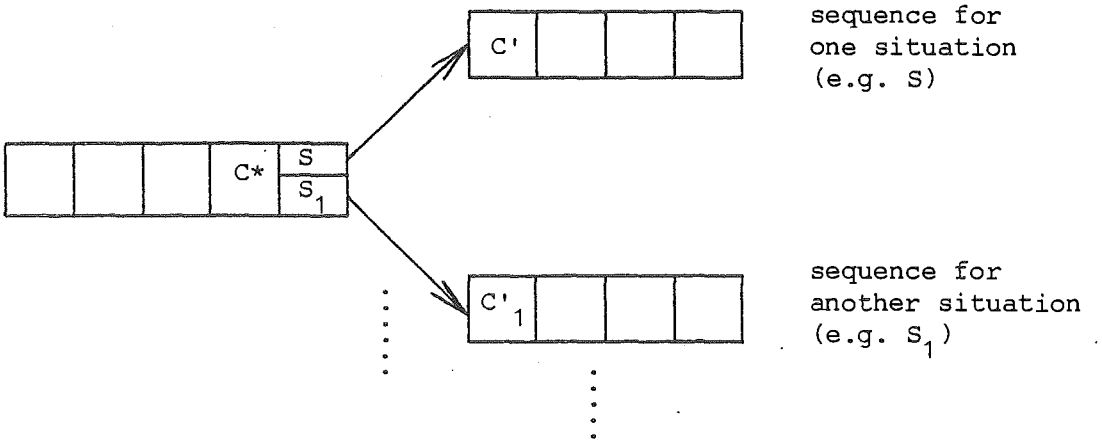
(b) Playback. A previously recorded sequence of motor commands is sent to the robot's arm control system (ACS). For example, as shown in the Figure, when the motor command +1 is sent to the ACS a movement of +1 cm/s/s might be produced. However, as explained in the text, if the ACS does not automatically compensate for all the forces opposing its actuators during the execution of +1, then +1 cm/s/s may not be the movement produced.



process of leading enables only the one motor command C---recorded when movement M is led--to be taught for producing M. However, a particular ACS will not always produce that movement M when sent that motor command C. Forces, say F', which oppose the ACS's actuators, may cause a different movement to be produced when C is sent to the ACS. For example, (a) the inertial forces exerted by the arm links and the load, and (b) gravitational forces, will oppose movements. C may produce M when the opposing forces are F. That is, the ACS "automatically compensates" for the forces F, but not for the forces F'. A motor command C' may produce M when C does not. For example, the motor command +2 may produce the movement +1 cm/s/s, when the motor command +1 does not. However +1 is recorded when +1 cm/s/s is led. So at times a teacher is forced to edit, via a keyboard or buttons, the motor command sequence he has led.

The second limitation of the popular implementation of the leading method is that the teacher must explicitly program any conditional branches that are required (see Takahashi & Kohno, 1982; Nagel, 1983; and McLaughlin, 1982). Conditional branches are required in the internal representation of a task if the motor commands the robot performs are to depend on sensory information. For example, as shown in Figure III-3, the motor command C' may be required when the sensory conditions are S, but C'_1 may be required when the sensory conditions are S_1. The teacher programs the branches by pushing keys to identify both the signal to branch on---say S versus S'---and the motor commands---say C or C'---to perform (Bertino et al, 1980; Taylor et al, 1982; Bonner & Shin, 1982; Cunningham, 1979). The more conditional branches that are required, the more the operator programs the robot and the less he leads it. He is more and more required to make

Figure III-3 Branching The teacher must explicitly program branches when using popular leading, if different motor commands are required in different situations. In the Figure, C' is the motor command that will be executed after C* if the sensory conditions are S, after C* is performed. If the sensory conditions are S_1, then C'_1 will be performed after C*. To explicitly program the conditional branch the teacher must identify S, S_1 and which motor command, C' or C'_1, to perform after each condition.



explicit programs for a task, rather than using his natural ability at performing the task.

Nagel (1983) has suggested that humans' problems in using programming languages to teach industrial robots, will be eliminated by experience. This objection to my avoidance of programming languages was dealt with in section 3.2 of Chapter II. Briefly, explicit programming of real world robots is difficult by nature.

Both limitations of popular leading require the teacher to do explicit programming, for editing motor commands and for forming conditional branches. Anyone unable to do explicit programming could not teach the robot, if motor command corrections or conditional branches were required. The main disadvantage of the explicit programming of robots is that the teacher must be both a programming expert and an expert at designing algorithms for robot tasks (McLaughlin, 1982; Lozano-Perez, 1983). Both my paths of improvement would enable editing of motor commands and teaching of conditional

branches without the need for explicit programming. The paths would therefore make led robots more teachable.

I shall now briefly map out the two proposed improvement paths, shown in Figure III-1, and then briefly introduce the steps along each path. The main difference between paths 1 and 2 is that the improvements along path 1 have a sequence of motor commands in their internal representations, while those along path 2 have a sequence of goals. This means that the improved methods on path 1 would be suitable for teaching a robot to perform a sequence of movements, while those on path 2 would be suitable for teaching a robot to achieve a sequence of goals.

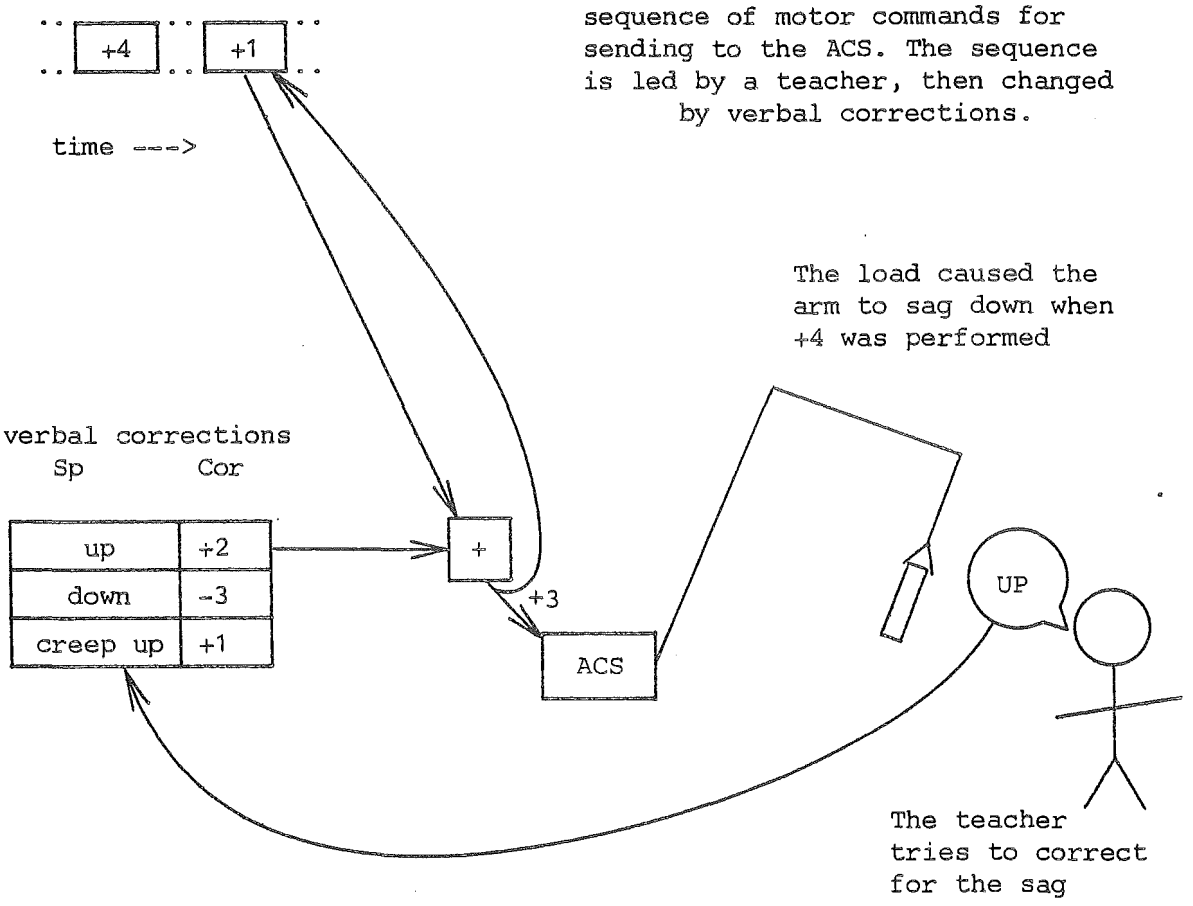
The first proposed path of improvement (i) adds a scheme called "verbal correcting" (VC) to popular leading, and then (ii) adds to that a "production system of corrections" (PSC). It will be argued that VC would enable a teacher to verbally correct a robot's motor commands while the robot performs movements. A motor command C, recorded when the movement M was led, could be verbally corrected to C', a motor command that produces M when sent to the ACS. A PSC would enable a teacher to use VC to teach conditional corrections, forming conditional branches in the motor command sequences, instead of having to explicitly program them in. These two steps complete the first path of improvement. I expect a led robot with VC and a PSC to be suitable for performing tasks which require one particular sequence of movements to be performed, but which require different motor command corrections in different situations, for producing a particular movement. Explicit programming should not be required for teaching such a robot these tasks.

The second proposed path of improvement (i) replaces the internal

representation provided by the popular leading method with a system called a "goal-seeking" (GS) system, and (ii) adds to that VC. It will be explained that a GS system enables a teacher to (a) set goals for the robot to achieve, and (b) teach the robot individual motor commands for producing individual movements. A GS system enables the robot to sense conditions and select a sequence of taught motor commands for achieving the taught goals. The robot would be more adaptable than one using popular leading because it could select individual motor commands which formed a path to a goal, even when that path had not been taught. The teacher need not explicitly program in all the possible conditions and sequences for reaching the goal. VC in a GS system enables a teacher to correct individual motor commands, by correcting the robot while it performs movements. I expect a led robot with VC and a GS system to be suitable for tasks which (a) require corrections to led motor commands, and (b) require the robot to achieve a number of goals in succession, but which do not require the robot to perform one specific sequence of movements in achieving a goal. Again, explicit programming would not be required for teaching such a robot these tasks.

The difference between VC, a PSC and a GS system is in the internal representations they provide. The internal representation provided by popular leading is an ordered sequence of motor commands, say $C_1\ C_2\ C_3\ \ldots\ C_i\ \ldots$, for example +1 +2 +2, as depicted in Figure III-2. VC adds to this internal representation an unordered list of two-tuples, that is pairs, of the form <speech words, motor command correction>. For example, "up +2" is a two-tuple that causes the current motor command to have +2 added to it. Figure III-4 depicts VC. A teacher can use spoken corrections rather than explicit programming

Figure III-4. Verbal correcting (VC) VC is explained in sections 1 and 2. In the example in this Figure the teacher said "up" when the robot executed an incorrect motor command, +4. This causes a correction +2 to be made to the motor command +1, which is the motor command executed immediately after "up" is recognized and the correction +2 selected. A +3 motor command, +1 corrected by +2, is executed and remembered in place of the +1. Thus the robot will send +3 to the ACS next time it reaches that point in the motor command sequence. The delay between the incorrect motor command +4 and the correction +2 should be about one second, as explained in section 1 (see also note 3). When the onset of error is rapid the teacher may learn to anticipate errors (Marteniuk, 1976), enabling him to correct the errors despite the delay. The verbal corrections are taught during a correction training mode, as explained in sections 1 and 2. The motor commands for one dimension are shown. The system depicted in the Figure comprises (a) an ordered list of motor commands, and (b) a list of two-tuples, <Sp, Cor>, where Sp means "speech words" and Cor means "correction motor command".



techniques, for changing the stored motor commands.

A PSC adds to VC an unordered list of motor command corrections for different robot conditions. The list of corrections is a list of three-tuples which have the form $\langle step_i , Cond_j , C'_k \rangle$, where C'_k is the corrected motor command to be executed on $step_i$ under robot conditions $Cond_j$. Figure III-5 depicts a PSC. The figure shows only one dimension of the position condition of the arm. A more comprehensive example will be given in section 3. The top three-tuple in Figure III-5, "3, 0 kg 5 cm, +1", gives a correction of +1 for the motor command on step 3, if the robot's wrist senses 1 kg and the position of the arm is 5 cm in the dimension shown in the figure. A PSC forms these conditional corrections as the teacher verbally corrects the robot. No explicit programming of such conditional branches is required. The three-tuples may also be called "production rules" (Nilsson, 1980), since they have the IF ... THEN DO ... form:

IF step number is $step_i$ AND robot conditions are $Cond_j$
THEN DO motor command C'_k.

Further explanation is given in section 3, of what constitutues a production rule. I shall call production rules just "productions".

A GS system provides an internal representation which comprises (a) an ordered sequence of goals, and (b) a network of productions. GS productions are different from PSC ones. As depicted by the example in Figure III-6, GS productions have the form,

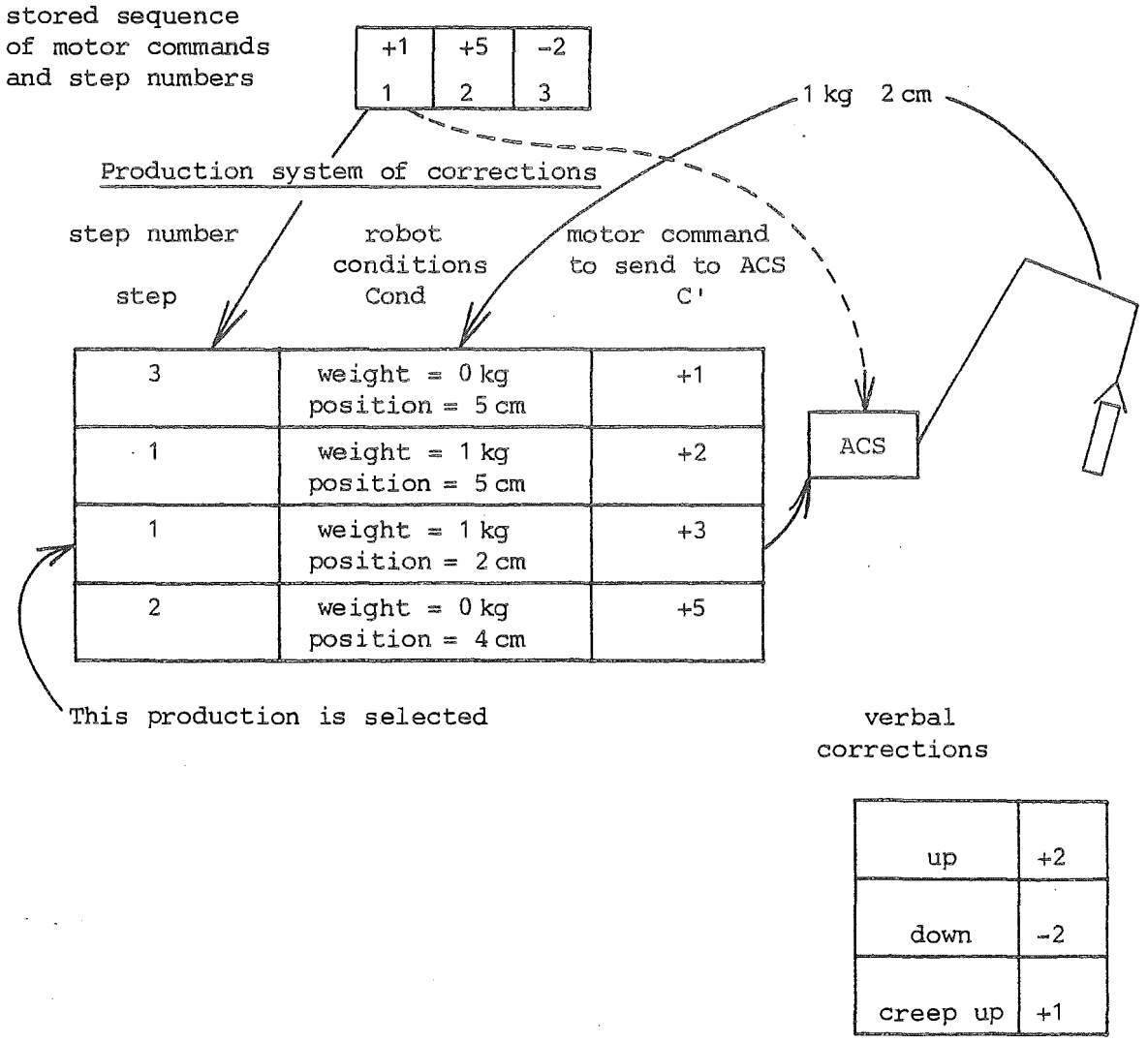
$$\begin{array}{llll} & & & Goal_{10} \\ & Cond_7 , prob\ (Cond_7 \mid 2,6) & & Goal_1 \\ Cond_2 , C'_6 , & Cond_2 , prob\ (Cond_2 \mid 2,6) , & Goal_3 , & Val \\ & Cond_4 , prob\ (Cond_4 \mid 2,6) & & Goal_8 \end{array}$$

Figure III-5 Production system of corrections (PSC) PSCs are explained in detail in sections 1 and 3. A PSC provides a corrected motor command, given the current robot conditions and the current step number. If there is no production for the step number in the sequence, under the current robot conditions, then the motor command in the sequence is sent to the ACS, as shown by the dotted line in the Figure. The system depicted in the Figure comprises: (i) an ordered list of motor commands; (ii) a list of three-tuples, or productions, <step, Cond, C'>, where step is the step number stored during leading, C' is a motor command and Cond is a set of robot conditions; and (iii) a list of two-tuples for VC.

The Figure shows how the motor command +3 is selected from the PSC. The step number is 1, +1 is the motor command in the sequence and 1 kg 2 cm are the current robot conditions. Thus the production indicated in the Figure is selected and +3 is executed. The actual formation of productions is shown in Figure III-9.

The dotted line shows +1 being sent to the ACS. This would happen if there was no production for +1 under the conditions 1 kg 2 cm.

Only the weight on the wrist and one dimension of the arm position are shown in the conditions in the Figure. Only motor commands for one dimension are shown.



or <$Cond_j$, C'_i, <Next $Cond_k$, prob(Next $Cond_k$ | j,i)>, <$Goal_\ell$>, Val>
where $Cond_j$ (e.g. $Cond_2$) are the robot conditions and C'_i (e.g. C'_6) is a motor command that will result in the robot conditions changing to one of the Next $Cond_k$s with estimated probability prob(Next $Cond_k$ | j,i) (e.g.

$Cond_7$, with an estimated probability , prob($Cond_7$ | 2,6),

$Cond_2$, with an estimated probability , prob($Cond_2$ | 2,6),

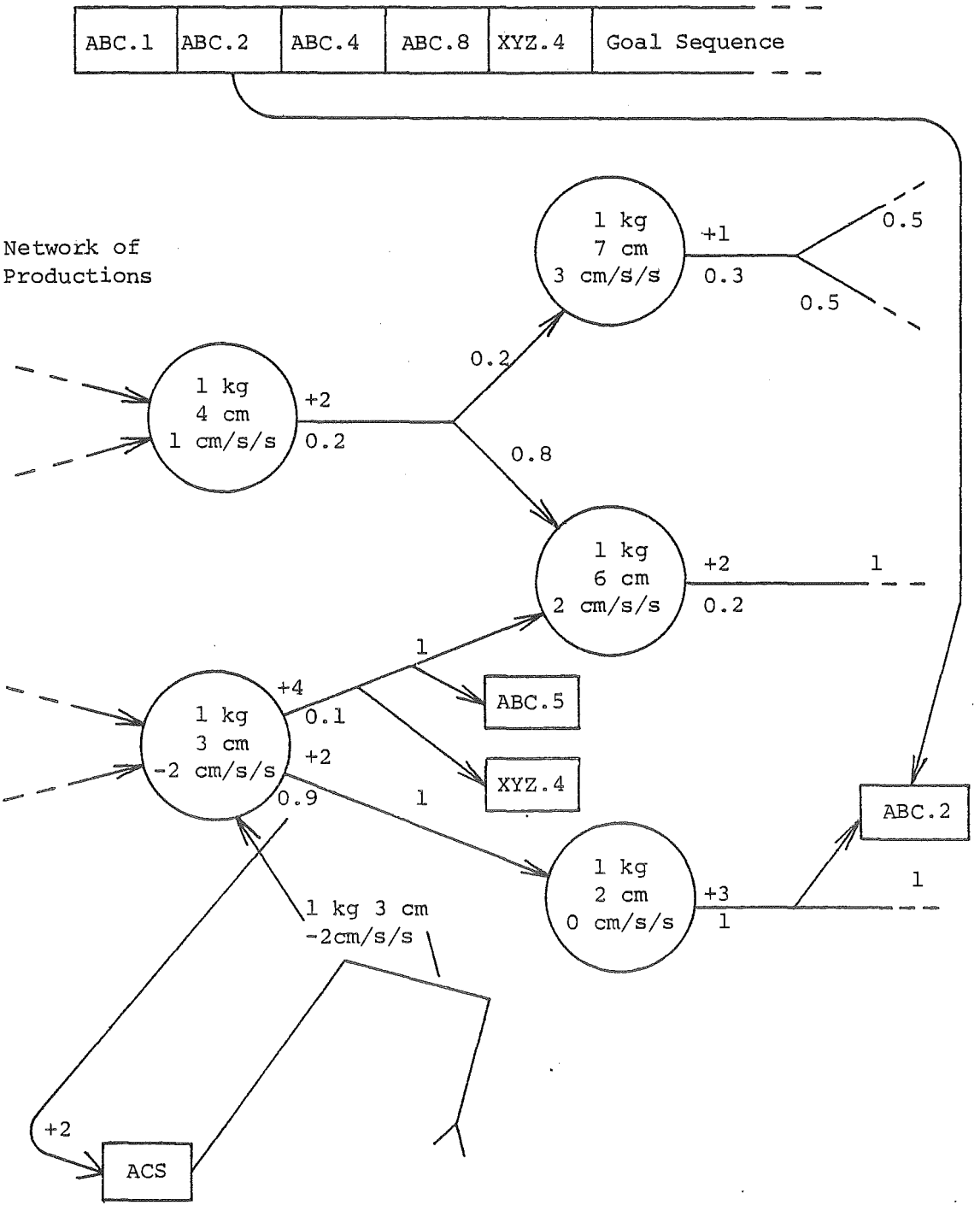
$Cond_4$, with an estimated probability , prob($Cond_4$ | 2,6).)

The $Goal_\ell$s are the goal codes from the sequence. The goal point for each $Goal_\ell$ is the production with that $Goal_\ell$ code in it. (e.g. $Goal_{10}$, $Goal_1$, $Goal_3$ and $Goal_8$ might all be to perform C'_6 under $Cond_2$.)

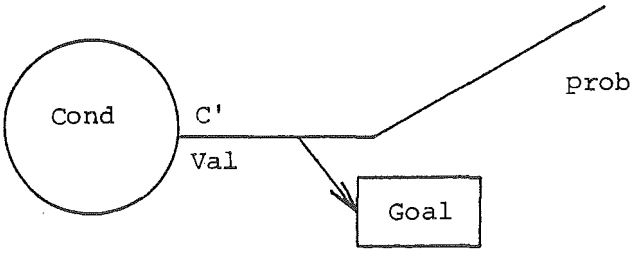
Val is an estimate of how worthwhile it is for the robot to reach $Cond_j$ and perform C'_i, given that it must reach the currently active goal. In Figure III-6 a production is depicted by a single path out of a Cond. A Val, motor command pair is given for each such path. A Cond is depicted as a circle. Goals are depicted as boxes. One or more estimated probabilities are shown for each path out of a Cond. When the active goal from the sequence is, say, $Goal_x$, the $Goal_x$ point in the network becomes the goal of the GS system. A path through the network from the current robot Cond to $Goal_x$ will be sought. For example, Figure III-6 depicts the robot's current conditions as 1 kg, 3 cm, -2 cm/s/s, and the active goal as ABC.2. The robot chooses the motor command +2, rather than +4, since the Val for +2, 0.9, is higher than 0.1, the Val for +4. This Val difference reflects the shortness of the path to goal ABC.2 via +2 to 1 kg, 2 cm, 0 cm/s/s, and +3, while +4 leads to no such short path to ABC.2. A GS system forms its network of conditional branches as the robot is led through movements and as the robot performs movements and is verbally corrected. Explicit

Figure III-6 Goal-seeking (GS) System A goal-seeking system is a network of productions, or quintuples, plus a goal sequence. The figure shows the motor command +2 being selected, since the robot conditions are 1 kg, 3 cm, -2 cm/s/s, and since +2 puts the robot on a short path to the active goal, ABC.2. The format for representing productions in the figure is shown at the bottom right.

Figure III-6 (caption opposite)



Format of Productions



programming is not required for forming conditional pathways to goals.

VC, PSCs and GS systems are discussed in sections 1 through to 4.

The two paths of improvement would increase the teachability and adaptability of led robots. VC should increase the teachability by enabling a teacher to correct motor commands using his own natural ability at verbally correcting humans. A PSC and a GS system should increase the teachability by enabling a teacher to teach conditional branches using his natural leading and verbal correcting abilities and his natural knowledge of task goals; a PSC for sequential tasks and a GS system for goal tasks.

A PSC and a GS system should increase the adaptability of led robots by enabling a robot to select its own motor commands, given the sensed robot conditions. In a PSC motor commands are selected for performing a sequence of movements. In a GS system motor commands are selected for achieving goals.

Section 1 lays out the development of the two paths of improvement. Section 2 (a) explains the proposed implementation of VC in more detail than does section 1, (b) gives an example ACS for which VC might be used, and (c) establishes the convergence and stability of VC under reasonable conditions. Section 3 describes PSCs in detail and gives a simple but real world example of a PSC. Section 4 (a) explains GS systems in detail, (b) gives an example task for a GS system, (c) explains the storage of productions during leading and execution, and (d) discusses the dual control nature of a GS system's motor command selection.

It will be explained in chapter IV that a GS system enables a robot to be taught a specific sequence of movements only by the teacher setting a goal after every movement in the sequence. I expect this to

be impractical for a human teacher. In chapter IV the second path of improvement is taken further by replacing the GS system with a multiple context learning system (MCLS). It will be shown in chapter IV that an MCLS enables both sequences and goals to be taught.

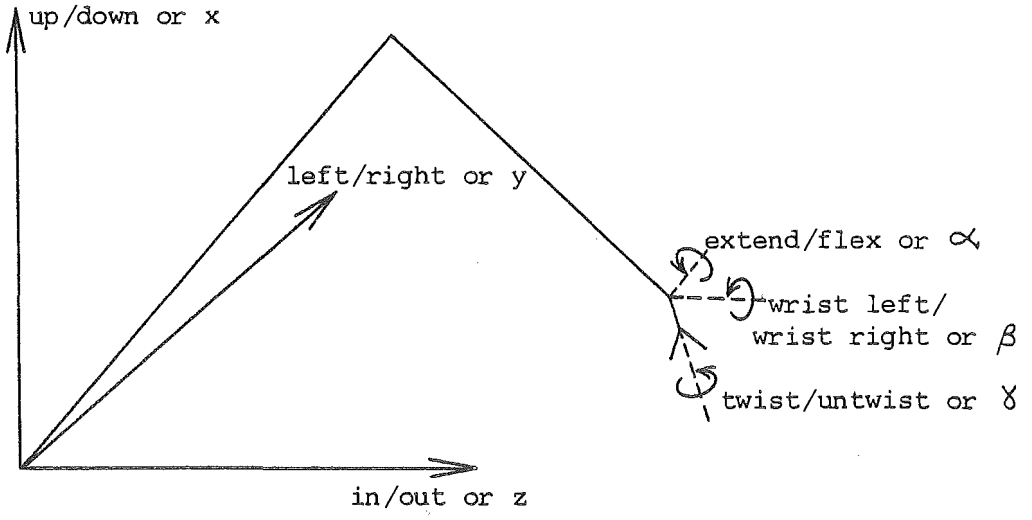
III.1 TWO PATHS OF IMPROVEMENT

Leading makes use of a teacher's natural ability at performing movements, and so enables the robot's teacher to be a human skilled in the task. The teacher need not be skilled in robot programming languages. So it is important that any improvements to the leading method also enable the teaching to be done by a human that is skilled in the task, but not skilled in robot programming languages. Also, it must be easy for the teacher to specify how a robot uses its abilities for performing a particular task, if those abilities are to be realized (Taylor et al, 1982).

I shall now introduce each improvement shown in Figure III-1. Path 1 in Figure III-1 will be introduced first. VC would enable a teacher to make additive corrections to stored motor commands using his own natural abilities. Figure III-7 lists a proposed sample repertoire of verbal corrections. The various size and direction commands are based on a set of verbal commands used for a real robot arm (Sachs & Leifer, 1979). In addition, the words "larger" and "smaller" are preprogrammed to scale up and scale down, respectively, the size of one set of corrections. For example, if the teacher says "smaller creep up" then two things would happen, if the robot had the verbal correction repertoire shown in Figure III-7. Firstly, because of the word "smaller" the three "up" corrections, +1 for "creep up", +2 for

Figure III-7 A proposed sample repertoire of verbal corrections.

(a) Coordinates



(b) A full repertoire of verbal corrections. The numbers shown on the right in the table on the next page are initial corrections, in motor command, that are made when the words on the left are heard by the robot. For example, Figure III-5 shows a correction +2 being made to the motor command +1, after the teacher says "up".

Smaller When the teacher says the word "smaller", before one of the corrections on the next page, the three corrections for the last word said are halved in size, before the requested correction is applied. For example, "smaller creep up" would cause the three up corrections to change from +10, +2, +1, to +5, +1, +0.5, and then the new "creep up" correction, +0.5, to be made. A coordinate system x,y,z,∝,β,ϒ for the corrections is shown in (a) above.

Larger "Larger" before a correction causes the three corrections for the last word to be doubled in size, and the new correction to be made.

(continued on next page)

Correction words	example correction in motor command units
zoom up	+10 in the x direction
up	+2 "
creep up	+1 "
zoom down	-8 in the x direction
down	-3 "
creep down	-0.5 "
zoom left	+12 in the y direction
left	+6 "
creep left	+2 "
zoom right	-7 in the y direction
right	-3 "
creep right	-1 "
zoom in	+8 in the z direction
in	+3 "
creep in	+1 "
zoom out	-10 in the z direction
out	-7 "
creep out	-1 "
zoom extend	+4 in the α direction
extend	+2 "
creep extend	+1 "
zoom flex	-3 in the α direction
flex	-2 "
creep flex	-0.5 "
zoom wrist left	+4 in the β direction
wrist left	+2 "
creep wrist left	+1 "
zoom wrist right	-5 in the β direction
wrist right	-3 "
creep wrist right	-1 "
zoom twist	+6 in the γ direction
twist	+3 "
creep twist	+2 "
zoom untwist	-5 in the γ direction
untwist	-4 "
creep untwist	-3 "

Figure III-8 Strategy for verbally correcting a robot using the repertoire shown in Figure III-7.

1 Correct large errors before correcting smaller ones. Start by scaling up verbal corrections with "larger", until large enough corrections can be made. Then scale down corrections with "smaller" as the size of error decreases.

2 Correct errors that are early in a sequence before errors that are later in a sequence. In doing this, restart the robot at the start of the sequence whenever large errors accumulate.

"up" and +10 for "zoom up", would be halved in size to +0.5 for "creep up", +1 for "up", and +5 for "zoom up". Secondly, the halved "creep up" correction, +0.5, would be made to the movement performed after the "smaller creep up" was recognized by the robot.

Corrections would be taught by the teacher pushing a button to put the robot into a special correction training mode, and then repeatedly, (i) saying a word or a few words (e.g. "creep up"), and (ii) leading a movement (e.g. an acceleration change of +1 cm/s/s in the up/down, or x, direction of Figure III-7). A two-tuple, <Sp, Cor> (e.g. <creep up, +1,0,0,0,0,0> where the correction is in the coordinates $x,y,z,\alpha,\beta,\gamma$) is recorded each time words are said and a movement is led. The teacher uses his natural ability at making movements to teach the robot motor command corrections.

When a verbal correction is made to the robot while it performs movements, I envisage this sequence of events occurring (see Figure III-4):

1 an incorrect motor command is executed (+4 in Figure III-4)

2 the teacher notices the onset of an error in the robot's movement

3 the teacher says words that have been taught as a correction ("up")

4 the robot recognizes the words

5 the robot makes the correction (+2 in Figure III-4) to the next motor command executed (+1) after step 4.

I expect this proposed VC scheme to be worth developing for led robots because:

(a) a teacher may use his own natural ability at verbally correcting other humans, as long as the robot does not exceed natural human correcting speed;

(b) a human may anticipate errors in movements (Marteniuk, 1976),

enabling him to overcome the delay between 1 and 5 above, when the onset of the error is rapid. The delay should be about one second (see note 3);

(c) a repertoire such as the one shown in Figure III-7, and a strategy such as the one shown in Figure III-8, may enable a teacher to correct most errors, in no more than 15 successive corrections. In the appendix is given a simple BASIC program which enables one to make additive corrections to 3 numbers in succession. This simple "trajectory" correction experiment indicates that huge errors may be corrected in less than 15 successive corrections, with the corrections of Figure III-7, and the strategy of Figure III-8. Small corrections can be made in fewer steps. An example run with the program is given in the appendix;

(d) if he likes, the teacher can stop the robot, put it into correction training mode, and teach new corrections. The new corrections may make the VC of a particular task easier and quicker;

(e) current speech recognition techniques enable short phrases from a limited vocabulary to be recognized (Sachs & Leifer, 1979; Simons, 1980; Nitzan, 1979; Flanagan, 1981; 1982);

(f) I establish the theoretical convergence and stability of VC in this chapter; and

(g) corrections to motor commands should also be able to make up for errors in the led movements.

In section 2 the details of VC are explained.

A correction to a motor command is good only for certain conditions; those conditions in which the particular correction counteracts the opposing forces that are not counteracted by the ACS. Under other conditions, a different correction may be required. If VC alone were

added to the popular leading method, then the teacher would have to explicitly program conditional branches when different corrections were required to the same motor command, under different conditions. As explained at the start of the chapter, Figure III-3 shows a conditional branch. A PSC would enable a teacher to teach a conditional branch simply by verbally correcting a motor command in each of the different conditions. The robot remembers the step number (step), conditions (Cond), and corrected motor command (C') in a three-tuple <step, Cond, C'>. For example, say the motor command after C* in Figure III-3 is C. Suppose C* is on step 4. When the conditions after step 4 are S, the teacher verbally corrects the robot, causing C to be corrected to C'. When the conditions after step 4 are S_1, the teacher again verbally corrects the robot, causing C to be corrected to C'_1. The robot remembers the three-tuples <5, S, C'>, and <5, S_1, C'_1>. So on step 5 a motor command (C' or C_1') will be selected conditionally on the robot's conditions (S or S_1). Figures III-5 and III-9 show the operation of a PSC. These Figures will be explained in more detail in section 3. Briefly: Figure III-5 shows how the robot selects a corrected motor command (+3), for step 1, having sensed the robot conditions (1 kg 2 cm); Figure III-9(a) shows how productions are formed by VC; and Figure III-9(b) shows how VC can change the internal representation. I expect a PSC to enable a sequence of movements, which requires different corrections in different situations, to be taught without the teacher having to explicitly program conditional branches. For this to be so, both (a) VC itself must work, and (b) during any such task the robot performs, the conditions the robot senses must distinguish the situations in which different corrections are required to the same motor command. Then the teacher's VC will

Figure III-9 The formation and changing of productions in a PSC. As in Figure III-5, only the weight on the wrist and one dimension of the arm position are shown, and only one dimension is shown for the motor commands.

(a) The formation of a new production is shown by the dotted lines. If a production with the same conditions and same step number already existed then the right hand side motor command would be changed to the new value, in the way shown in (b) on the next page.

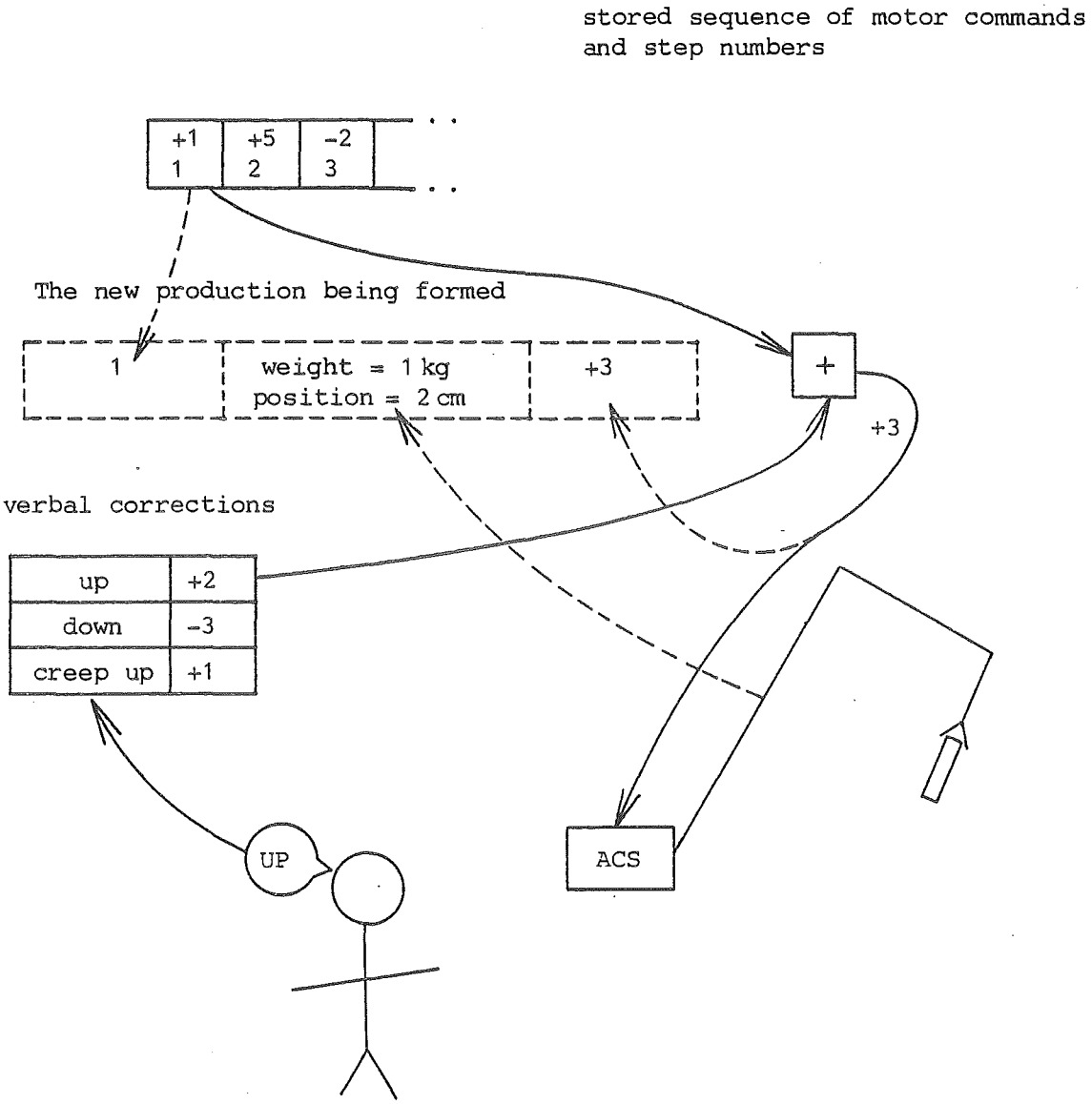
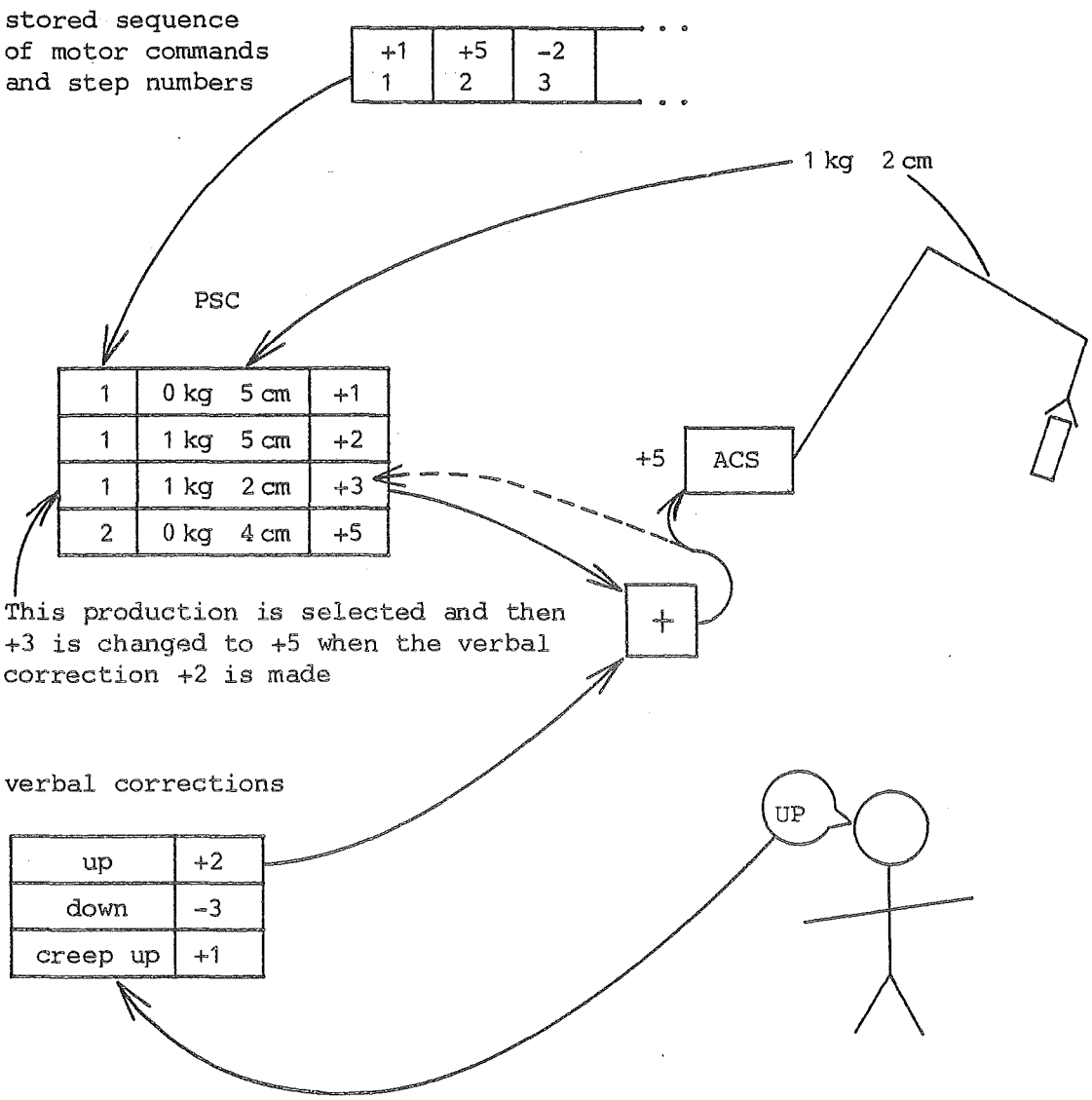


Figure III-9 (continued)

(b) Verbal corrections can change the motor commands on the right hand sides of productions, as shown by the dotted line.



form the necessary conditional corrections in the PSC. In section 3 the details of PSCs are explained. A slightly more advanced PSC is presented in the appendix.

My first path of proposed improvement to the leading method ends with the addition of a PSC to VC. The second path will now be introduced. It begins by replacing the sequence of motor commands, used in the internal representations throughout path 1, with a sequence of goals, and adding an unordered list of productions. The unordered list of productions, along with the sequence of goals, is a GS system. The internal representation provided by a GS system is more suited than those provided in path 1, for a task that requires a goal or goals to be achieved, but not particular sequences to be performed. In different situations the movement sequences that achieve the goals may be different. A movement M may be required in one situation for reaching a goal. A movement M' may be required in another situation for reaching the same goal. To use a method from improvement path 1, one with motor command sequences in its internal representation, a teacher might need to teach many different sequences for achieving the goal under many different conditions. In a system without a PSC, the teacher would need to explicitly program conditional branches into the internal representation, since different motor commands would be required in different situations. With a PSC in the robot, VC could be used for teaching different motor commands for different situations. However, VC with a PSC would be practically useful only for making corrections to a led sequence, not for forming completely <u>different</u> sequences of movements for different situations. The different sequences required for reaching the goal could be of different lengths and consist of quite different movements. VC would be an extremely

clumsy and tedious way of turning one sequence of motor commands into another quite different sequence. So if various ways of getting to a goal must be taught, then some form of explicit programming would be required in a VC-plus-PSC system. Branches could then be programmed into the sequence of motor commands shown in Figures III-4, III-5 and III-9, in the way shown in Figure III-3. Quite different paths to a goal could be led and corrected, using a PSC and explicit programming of conditional branches.

A GS system, shown in Figures III-6 and III-10 and briefly explained near the end of the last section, enables a teacher to teach different ways of getting to a goal without him having to do explicit programming, nor difficult verbal correcting. The teacher (a) sets a goal or sequence of goals for the robot, and (b) teaches individual motor commands. The robot senses robot conditions, and selects its own motor commands for achieving each goal in turn. For example, near the end of the last section I explained that +2 would be selected under conditions 1 kg, 3 cm, -2 cm/s/s, if the active goal in Figure III-6 were ABC.2. Figure III-10 will be explained in section 4. Note that in Figure III-10 a production is represented by a rectangular box, rather than as a path out of a circle, or Cond, in Figure III-6. Also, the productions shown in Figures III-10(a) through to (c) and (e) are simplified in that each production has only one "Next Cond," and at most one goal. In addition, there are no estimated probabilities shown. The simplification is to aid my discussion of GS systems. Briefly:

Figure III-10(a) shows how a motor command is selected,

Figure III-10(b) shows how quintuples are formed during leading. No force information is stored during leading because the forces that

Figure III-10 Goal-seeking (GS) system A goal sequence is used to activate goals in the GS system. A GS system may be described by:

(i) an ordered list of goals;

(ii) a list of quintuples,
<Cond, C', <Next Cond, prob(Next Cond)>, <Goal>, Val>,
where Cond is a set of conditions of the robot, C' is the motor command to perform, Goal is a goal, Next Cond is the conditions which will occur with probability prob(Next Cond), when C' is performed under Cond, and Val is a measure of how many steps there are to the next goal, from the condition Cond, if C' is performed; and

(iii) a list of two-tuples for VC.

More complex quintuples are described in the text and shown in Figure III-10 (d). The motor commands for one dimension are shown. A GS system is depicted also in Figure III-6.

(a) Selecting a motor command. The example values of quintuples shown in the Figure assume that a quintuple k actions from a goal production is worth $.9^k$. The formation of productions is depicted in (b). The effect of VC is shown in (e).

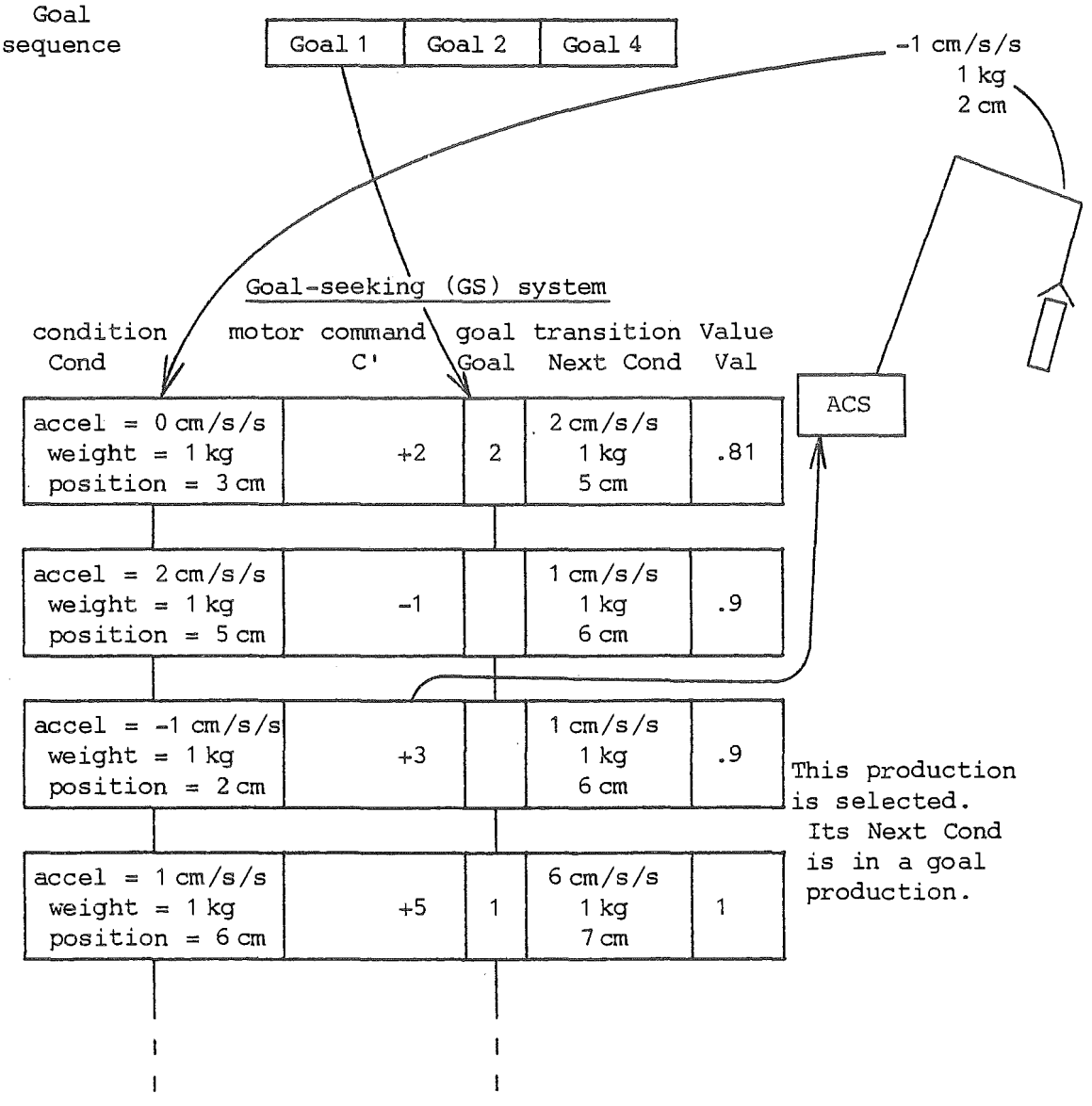


Figure III-10 (b) Leading forms productions which have non-force Conds in them. In the Figure a dotted box surrounds the production being formed during leading.

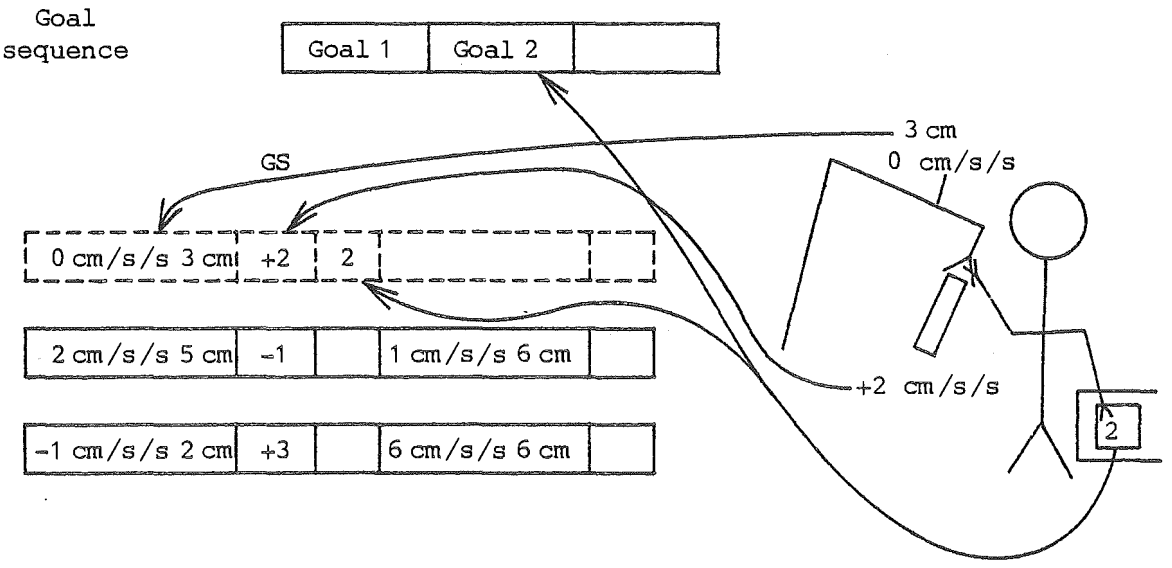


Figure III-10 (c) When the robot's non-force current conditions match the non-force Cond of a production, that production may be copied, and the current force conditions put into the Cond of the new production. The formation of the new production is shown by dotted lines. No copy is made if an existing production has the full Cond and the C'.

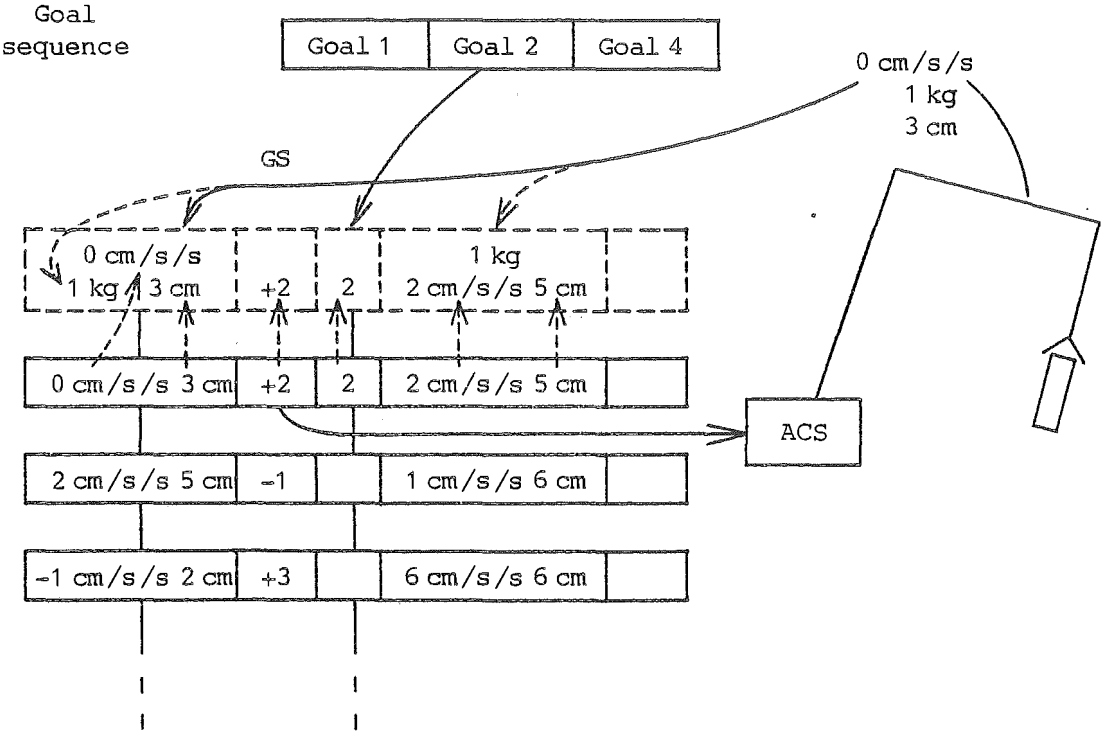


Figure III-10 (d) Example quintuples with more than one goal and with probabilistic transitions. The first three quintuples are the type formed during execution. The fourth quintuple is of the type formed during leading. The second production was made from the first four-tuple of the led production. The active goal is goal 2 in task ABC. So the Val of the third production is the maximum, 1. The value of the second production is .9, assumming that each step away from a goal causes a reduction by .9. The first production is then worth .8 x .81, since it is two steps away from the goal, with a probability of .8.

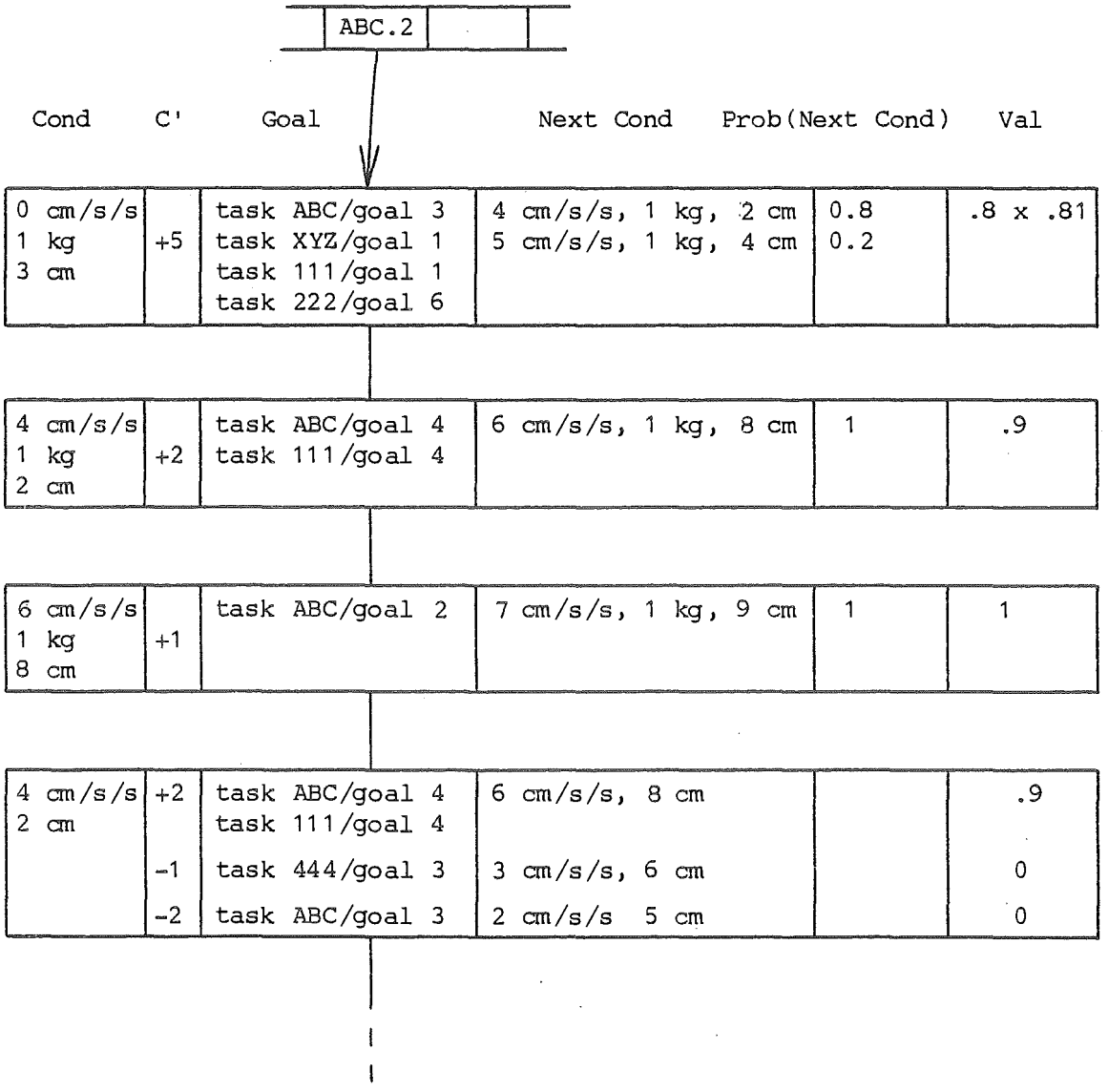
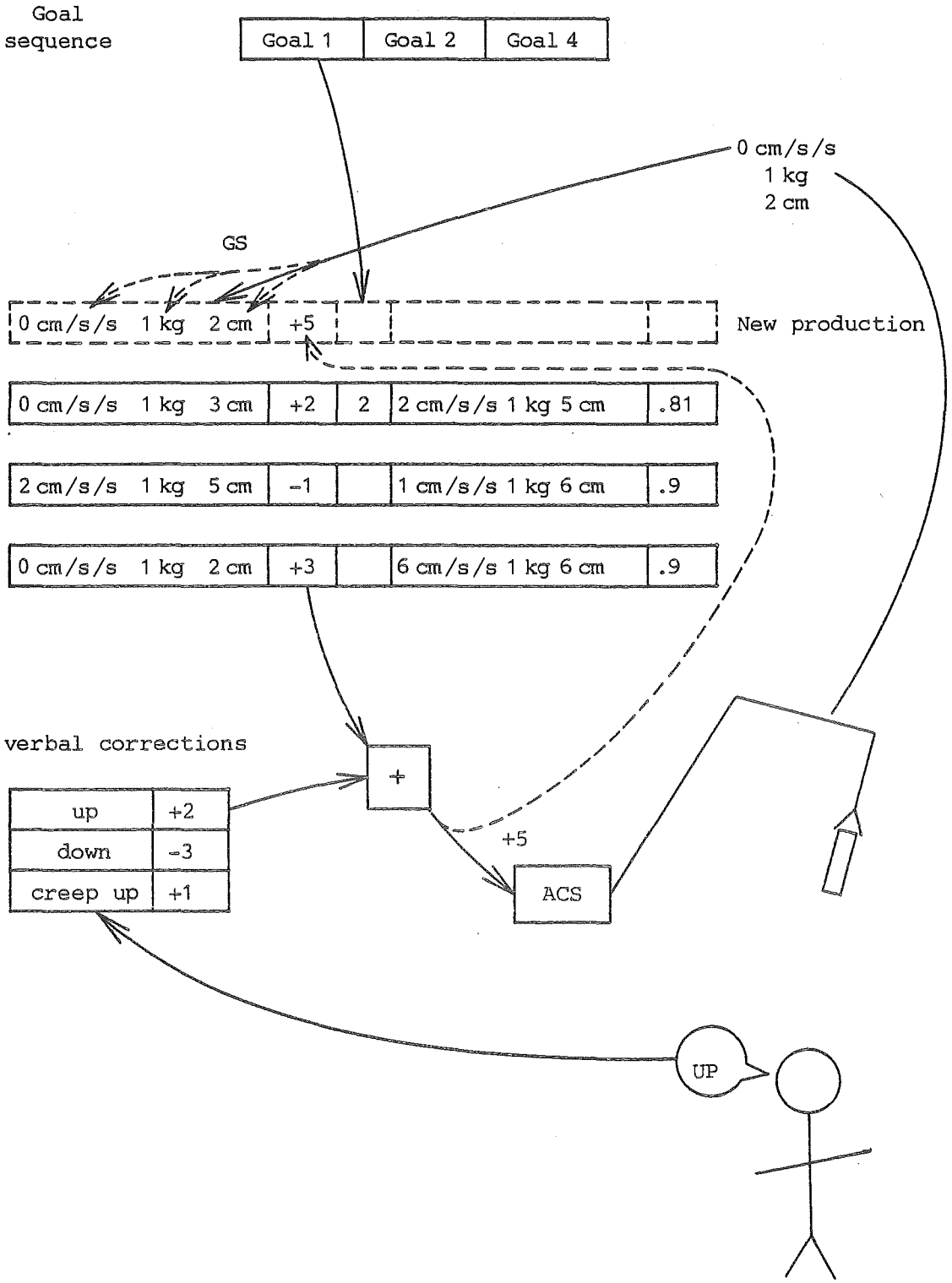


Figure III-10 (e) Verbal correcting forms new productions in the GS system, as shown by the dotted lines.



occur during leading are not the same as those that occur during execution. For example, a wrist weight sensor may register 0 kg while the teacher holds the robot's hand, even though there is a 1 kg weight in the hand.

Figure III-10(c) shows how force information is "filled-in" during execution.

Figure III-10(d) shows the more complex sort of quintuples that can occur in a GS system.

Briefly, the teacher leads the robot from its start position to a goal, pushes a "goal button", then leads it to another goal and pushes another goal button, and so on to the end of the task. [Further elaborations enabling a teacher to give task names, and to combine tasks together to form other tasks, are suggested in section 4.] The teacher then puts the robot into execution mode. The robot selects motor commands that cause condition (Cond) changes that take it from its current condition to the first goal that was set, then to the second goal set, and so on. If a set of conditions (Cond) occurs for which the robot has learned no motor command, then it will stop. The teacher could lead it through movements, teaching motor commands for that situation. VC might be employed with a GS system, as shown in Figure III-10(e). The corrections would cause new quintuples to be formed, which could enable a robot to reach a goal. Also, if the robot stopped, having reached a Cond for which it has no motor command, VC could be used to verbally command the robot---"up", "down", "left" etc---and cause productions to be stored.

In section 4 the details of GS systems are explained. In section 4.1 an example task for a robot with a GS system and VC is given. The task is a "die-casting transfer" task. The robot has to take objects

out of a die-casting machine and put them on a conveyor. There may be obstacles between the machine and the conveyor, so the sequence of movements required may vary from time to time. The task is taught as two goals. The first is to reach the machine without an object in the hand and grasp an object. The second is to reach the conveyor with an object in the hand and let go the object. The teacher teaches the die-casting transfer task by (a) leading the robot to pick up an object from the die-casting machine, (b) pushing a button for goal 1, (c) leading the robot to drop the object on the conveyor, and (d) pushing a button for goal 2.

Section 4.2 explains that in each set of conditions, Cond, a compromise is made between (a) performing an action most likely to put the robot on a path to the goal, and (b) performing an action whose prob(Next Cond)s are inaccurate, in case the action puts the robot on a better path to the goal. A near optimal compromise due to Cashin (1970) is discussed.

As will be reported in chapter V, the GS internal representation has been experimentally verified with a simple single degree of freedom arm. The experimental verification shows that GS quintuples can be recorded in real-time, during a teacher's leading, and employed during execution, so that a goal may be achieved. However, the PSC and VC have not been experimentally verified. There are two main reasons why the experimental verification of a GS system is important, while the experimental verification of a PSC and VC is not so important. Firstly a PSC and VC are additions to the proven stored sequence of motor commands used in popular leading. A GS system is a different internal representation from that used in popular leading. Secondly, the GS system forms the basis of improvement path 2, which is extended

to an MCLS in chapter IV. MCLSs are the subject of chapters VI, VII, VIII, and IX. However, in this thesis neither a PSC nor VC are extended further than in this chapter. I simply argue that a PSC and VC are worth developing for robots.

In chapter IV the second path of improvement is taken further by extending the GS system to an MCLS. An MCLS enables both goals and sequences to be taught. Firstly, an extension of a single GS system forms a "context system" which enables sequences to be taught. The extension is to have "contexts", rather than Conds in the productions. The contexts may contain the robot's recent motor commands and conditions, so that the robot can remember the sequence it is performing. For example, the robot is able to count its motor commands, and so perform a specific sequence of motor commands. Secondly, by having more than one context system in the MCLS---at least one of which is taught sequences and at least one of which seeks goals---the robot can have both goals and sequences at the same time.

In chapter IV I am not able to argue that a robot with an MCLS would be practically teachable by a human skilled in the task. I do suggest that the methods discussed in this chapter, III, may be practically useful. I expect the combinations listed below to be preferred to the popular leading method:

Improvement Path 1

(a) VC, because it may enable motor commands to be corrected by a human teacher using his own VC ability;

(b) VC with a PSC, because a PSC should enable a teacher to teach conditional corrections without having to explicitly program them;

Improvement Path 2

(a) a GS system, because it enables a teacher to set goals for performing tasks in which particular sequences are not required. The teacher does not have to explicitly program conditional branches in order to teach different sequences for getting to a goal;

(b) VC in a GS system, because it enables a teacher to teach new productions by verbally correcting the robot while it executes a task. The new productions may complete a path to a goal, enabling the robot to achieve the goal.

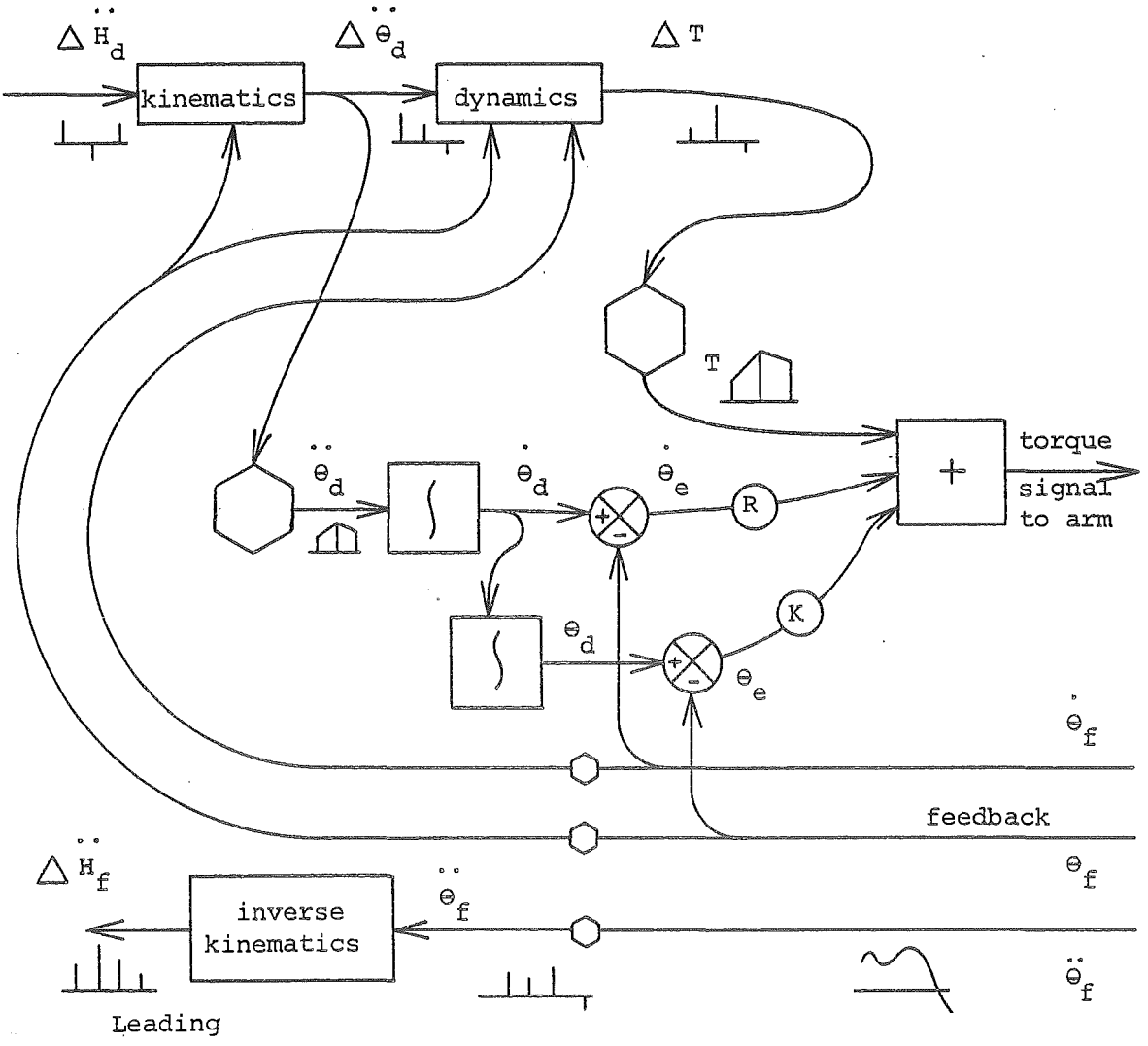
III.2 VERBAL CORRECTING (VC)

As explained at the beginning of this chapter, forces will oppose robot arm movements. Any particular ACS will automatically compensate for only some opposing forces. Consider an ACS that does not counteract the gravitational force on the arm's load. That is, the ACS does not automatically compensate for the load's weight. The robot might have to use a different motor command to move a large mass through an upward movement, from the motor command required for moving its unloaded arm through the same movement. The motor command acquired during the leading of the movement will be one for performing the movement without a load. So this motor command may need to be corrected, in order to achieve the desired movement with the load.

If the ACS compensates for the static effect of gravity then the motor command for lifting the unloaded arm through a movement would also lift the loaded arm through the same movement. However, different loads cause different inertial and joint interaction forces to be exerted on a robot arm (see Benati et al, 1980; Hollerbach, 1980; Raibert, 1978; Luh & Lin, 1982). Even if the ACS also compensated for these dynamic effects, there would still remain other types of force interaction that it would not compensate for. For example, the forces exerted on a robot's arms while it rode a bicycle would depend partly on the characteristics of the bicycle. Regardless of how much an ACS compensates for, there will still be other force interactions that it does not compensate for. Corrections, for these other opposing forces, must be made to the motor commands which are sent to the ACS, as explained at the start of this chapter.

Figure III-11 shows an example ACS. Actions are for hand movements.

Figure III-11 Arm control system (ACS) The symbols in the diagram are explained in the key. The control system applies torques to the joints of the arm. The joint torques depend on the desired hand acceleration changes and the joint position and velocity errors. Led joint movements are converted into hand acceleration changes and stored as actions. The dynamic model of the arm is fixed. It does not take into account the forces external objects exert on the arm.



KEY

H hand position (six dimensions)

Θ joint position (six dimensions)

T joint torques (six dimensions)

· velocity ·· acceleration

d desired value

f feedback value

e error; desired minus feedback

⌠ time integral

a discretely timed function

a continuous version of a discretely timed function

a continuous function

R velocity error gain

K position error gain

△ change in

An action is for a change in the acceleration of the hand. The hand acceleration is a six-dimensional vector which is the acceleration of the hand both in three-dimensional position space and in three-dimensional orientation space, as shown in Figure III-7(a). The actions are converted from hand coordinates into joint coordinates by a kinematic model of the arm. The kinematic model represents the geometry of the arm. The torque changes that must be made at each joint are calculated from the required joint acceleration changes using a dynamic model of the arm. The joint positions and velocities that correspond to the desired joint acceleration changes are subject to servo control. This servo control provides necessary stability against gravity (Benati et al, 1980), and counteracts minor disturbances not included in the models. For example, there may be varying amounts of friction and striction in the joints. A system similar to the one depicted in Figure III-11 has been suggested by Raibert & Horn (1978). Their system employs a tabular representation of the dynamic model of the arm. The dynamic and kinematic models' calculations can also be performed in real-time by a computer, such as a PDP 11 (Hollerbach, 1980). Control systems for arms are discussed in section 2.2 of chapter II, and in MacDonald (1981).

The internal representation this robot has of a task must be in terms of changes in the acceleration of the hand. Thus, during this arm's leading mode, changes in joint accelerations would have to be transformed into hand coordinates, and recorded. Suppose that led movements are recorded in a sequence of fixed length time intervals. The average changes in hand acceleration over an interval are recorded. These changes are played back to the ACS in Figure III-11 during the execution of the task, as depicted in Figure III-2(b).

Now, suppose that the dynamic model of the arm is a fixed one. It does not take into account the forces that external objects exert on the arm. For example, the arm might be slower when there is a heavy object in the robot's hand, or it may sag down. Tasks will often require a robot to move objects around and exert forces on objects in the environment. Some of the hand movement actions that are recorded during the leading of such a task must be replaced with other hand movement actions, so that the required movements are achieved.

Using VC the teacher may verbally correct the robot's movements. For example, an action for a change of +1 units along the up/down dimension might have been acquired during the leading of a task. The teacher might have led an acceleration change of +2 along the up/down dimension, with the sound "up", during the correction training. Suppose the teacher said "up" sometime before the robot performed the change of +1. Suppose the +2 correction was selected just before the performance of +1. The robot would change the +1 into +3. The +3 would be stored in place of the +1, as shown in Figure III-4. Next time the robot would perform +3 without the teacher saying "up". The teacher could have the robot perform the task several times, correcting the movements more and more each time.

The sounds might cause several actions to be produced in a row. Each action, in turn, would be added to consecutive actions the robot had already stored. For example, imagine that 3 consecutive actions the robot had previously stored were +20, +30, +40 units. Suppose the teacher said "up" before the robot performed +20; so that the correction for up began when +20 was performed. Suppose that he had previously led +10, +10, +10 after saying "up". Then the robot would perform changes of +30, +40, +50 in those 3 consecutive time intervals,

and record +30, +40, +50 for performing in those 3 time intervals of the task. "Cor" in the two-tuple would be the sequence of correction actions, +10 +10 +10.

The teacher's correcting sounds may not produce the movements he led with the sounds. However, the corrections will in general reduce the error that the teacher is trying to eliminate. The error can be successively reduced. The teacher can correct the error by applying one or more corrections, as explained in section 2.1, below. I establish in section 2.1 and the appendix that VC is stable and convergent under reasonable conditions.

Initially the minimum response time of a teacher's verbal corrections will limit the speed at which a robot can move and still have its movements corrected. A teacher must make a verbal correction to a movement very soon after a movement error is observed, so that the erroneous motor command is corrected. However once he has seen a robot perform a task incorrectly several times, a teacher may be able to make a verbal correction slightly in advance of the movement error in a particularly rapid movement. Humans verbally correct the behaviour of other humans, so I expect a human teacher to have little difficulty verbally correcting a robot, as long as the robot does not exceed natural human verbal correcting speed. Now, when a human knows just where an error occurs in a movement sequence, he is able to anticipate the error with his correction. So verbal corrections could be made to extremely fast robot movements.

Sometimes a teacher may lead a task incorrectly. Either (i) he may make an error in the movements he leads, or (ii) he may not be physically capable of leading the required movements. For example he may not be able to move the arm fast enough. Suppose the movement M

is required. The leading of movement M* will cause a motor command, say C*, to be recorded. VC may be used to correct C* to C', the motor command that produces M.

Of the total correction, C'-C*, part, C'-C, will be a correction for opposing forces not automatically counteracted by the ACS. The other part, C-C*, will be a correction for the error, M*-M, in the movement that was led. C is the motor command that would have been stored if M had been led. However, it will make little difference to a teacher what the source of an error is. If for any reason a stored motor command does not actually produce the movement the teacher wants, then he must correct that motor command. The correction of incorrectly led tasks is discussed in more detail in section 2.2.

I expect that a teacher using verbal corrections could correct most tasks that he could lead. The VC method enables any action that can be led to be used for correcting the task. Any action that can be expressed as the addition of leadable actions can be put into the internal representation of the task. The teacher directly corrects the movements, causing indirect corrections to be made to the actions in the internal representation. Some corrections cannot be made with VC. Recall that corrections are taught by the teacher leading movements, and saying sounds. For example, to teach a robot to make an upward correction when it hears "up", a teacher says "up" and leads an upward movement. However, some movements cannot be led. For example, a teacher might not be able to lift a robot's arm at all. He could not teach upward corrections. He could not make upward corrections. Corrections that cannot be made are discussed in section 2.3.

The general problem of speech recognition---machine recognition of normal, everyday human speech---has not yet been solved (Doddington &

Schalk, 1981; Flanagan, 1981; 1982). However, existing robots can recognize and respond to words and short phrases made up from a limited vocabulary (Sachs & Leifer, 1979; Simons, 1980; Nitzan, 1979; Flanagan, 1981; 1982).

III.2.1 Stability and Convergence of Verbal Correcting

A teacher will find it very difficult to know which actions are required for a movement task[4]. He does not know the internal representation of the task. Having seen the error in the movements a robot makes as a result of being led through a movement task, the teacher will find it difficult to know the replacement actions that are required. Arms are complex, dynamic and nonlinear.

However, the teacher will find it much easier to make a correction to the actual movements[5]. If the teacher can lead the robot through a task then he knows the movements required in the sense that he knows how to do the task himself. Even if he cannot lead the robot through the task he may well be able to see what corrections are needed to the robot's movements. The teacher does not have to know the actual actions that are required. The teacher leads correcting movements in a way that enables him to verbally command the execution of correcting actions. The correcting actions may not produce the correcting movements required when they are verbally invoked by the teacher during the robot's execution of the task. The additional accelerating forces will be subject to the opposing forces. However, the correcting actions may cause the error to be reduced since they cause forces to be exerted in opposition to the movement-opposing forces.

A teacher's successive corrections will naturally satisfy Dvoretzky's conditions (Dvoretzky, 1956; Wilde, 1964) for the

convergence of a stochastic approximation scheme, as long as the interaction between the robot and its environment is not too complex. For example, if another human is physically interacting with a robot's arm, a teacher might find it impossible to correct the robot's movements. The other human might exert forces on the robot's arm in a deliberate effort to thwart the teacher's VC.

Dvoretzsky's conditions and the way a teacher's verbal correcting satisfies them are discussed in detail in the appendix. Briefly, the teacher's correcting may converge because he may naturally adjust his corrections to ensure that (a) correction "overshoot" tends to decrease, (b) any increases in the movement error tend to decrease as successive corrections are applied, and (c) correcting does not stop until the movement error is reduced to an acceptably small size.

The error in a movement that is part way through a sequence of movements may be partly influenced by errors in previous movements. If there is an error in the movement performed at time t, then the state of the arm will be "wrong" at the start of the next movement, say at time t + T. Thus the error in the movement performed at t + T will be due both to the error in the movement performed at time t, and any error in the action for the movement at t + T.

The influence of previous movements' errors can be treated as random fluctuations added to a teacher's verbal corrections. As long as random fluctuations eventually vanish they do not prevent a stochastic approximation scheme being stable and convergent (Dvoretzky, 1956; see appendix).

I explain in the appendix how any sequence of movements may be formulated in a way that ensures the first movement is not affected by previous movements. This is summarized in the next paragraph. So the

first movement converges to the correct value because (a) a teacher may naturally adjust his corrections to a single movement so that it converges, as explained above, and (b) there are no movements before the first one that affect its error. Since the first movement converges, its influence on the error in the second movement vanishes. So the second movement converges, and so on. This is not to say that corrections to the kth movement of the sequence will be of no use until the 1st through k-1th movements have converged, but only that the kth movement may not finally converge until the 1st through k-1th movements have converged. By taking the robot through the task several times the teacher can gradually reduce the movement errors to within the limits required by the task.

Briefly, there is no stationary starting position for a task in which a robot performs a continuous cycle of movements. However, a teacher may provide a "lead-in" sequence to such a task, from a stationary starting position. Once the movements have been corrected the teacher may need to remove the lead-in.

III.2.2 Correcting Movements

If a teacher leads the correct sequence of movements then the only changes that may be required are corrections to the individual actions stored. No actions need to be inserted into the sequence, added on to the end of the sequence, or deleted from the sequence. If a teacher leads only the beginning of a sequence of movements, then actions must be added on, as well as the led actions being corrected.

VC enables both the correction of actions, and the adding of actions to the end of a stored sequence. The latter is really just the correcting of a sequence of null actions.

VC does not enable actions to be inserted into, or deleted from a sequence. The effect of insertion or deletion can be produced. The teacher must correct all the actions following the point in the sequence where the insertion or deletion is required. This could be very tedious. The closer a teacher's leading is to the required sequence of movements, the less inserting and deleting will be required. It may be better for a teacher to abandon leading movements, and use verbal corrections on null actions, if his leading is too bad. He verbally guides the robot through the task.

III.2.3 Corrections that cannot be made

Suppose a teacher cannot lift a robot's arm at all. He is unable to counteract the forces opposing his movement of the robot's arm. Then a correction to lift the robot's arm cannot be led by him during the correction training mode. So upward corrections could not be made during the execution of movements.

Some corrections that cannot be led as one movement can be produced by successive corrections. For example, suppose a correction of +10 is

needed to a movement performed by the robot with the ACS of Figure III-11. Suppose that a teacher cannot accelerate the robot arm at +10 cm/s/s, which is the movement that must be led in order to teach a +10 correction action. He could not teach a correction action of +10 during the correction training. He might be able to accelerate the arm at only +2 cm/s/s. Still, a correction of +10 could be produced by the teacher making 5 successive corrections of +2.

In general a correction can be made if there exist a finite number of leadable movements that add up to that correction. This is discussed in more detail in the appendix. Note that the words "smaller" and "larger" do not enable more corrections to be made than can be made using led corrections. They simply enable some corrections to be formed from a smaller number of single corrections. If a teacher is capable of leading a correction x (e.g. +2), then he is capable of leading a correction of one half x. A correction of 2x can be produced by making the correction x twice.

III.3 PRODUCTION SYSTEM OF CORRECTIONS (PSC)

As explained at the beginning of this chapter, a PSC comprises (a) an ordered sequence of motor commands, and (b) an unordered list of three-tuples $\langle step_i, Cond_j, C'_k \rangle$. The sequence of motor commands is stored during leading, just as it is for the popular leading method, in the way shown in Figure III-2(a). In addition, a sequence of step numbers is stored, one with each motor command, as shown in Figures III-5 and III-9. Initially, during execution a motor command from the stored sequence is sent to the ACS at each step (see the dotted line in Figure III-5).

As the teacher verbally corrects the robot, productions are formed, as shown in Figure III-9(a). To form a production the verbally corrected motor command is stored with the step number and the robot conditions. For example, in Figure III-9(a), the production "1, 1 kg, 2 cm, +3" is formed when the motor command +1 is corrected by +2 under conditions 1 kg, 2 cm on step 1. Once a production is formed for a step and set of conditions, the motor command given by that production will be sent to the ACS for that step and those conditions; for example +3 for step 1 and 1 kg, 2 cm. If a subsequent verbal correction is made to that step under those same conditions, the motor command will be changed, as shown in Figure III-9(b). The +3 is changed to +5 by a further correction of +2.

A more advanced PSC is explained in the appendix. It enables corrections learned on one step for producing a movement under a set of conditions, to be used on a different step for producing the same movement under the same conditions.

A production system comprises (a) a global database, (b) a set of

production rules, and (c) a control system (Nilsson, 1980). Each production rule has a precondition which is either satisfied or not satisfied by the database. In the case of a PSC, the database is the current robot conditions, Cond, and the step number, step. If the precondition is satisfied then the production rule may be applied. The control system chooses which of the applicable rules is in fact applied. In the case of a PSC, either one or no rules are applicable, since only one production rule with a particular step number and set of conditions can ever be formed. So the control system of a PSC is trivial; if a production rule is applicable then it will be applied. The "application" of a production rule involves just sending the motor command in the production rule to the ACS. I shall call production rules just "productions".

An example of a PSC for a robot with the ACS in Figure III-11 will now be given. A robot with the ACS of Figure III-11 might have been led through the movement that is an acceleration change of 10 cm/s/s upward, at, say, step 21, causing the action UP +10 to be recorded as an internal representation of that movement. The action UP +10 may have been corrected to UP +20 by VC. This would cause a production to be stored with 21 on the left hand side and UP +20 on the right hand side, in the way shown in Figure III-9(a). The UP +20 might produce the movement of 10 cm/s/s upward. The conditions, Cond, in this production might be:

HAND POSITION IS 30 cm, 40 cm, 40 cm,

HAND ORIENTATION IS 10 deg, 20 deg, 10 deg,

HAND VELOCITY IS 10 cm/s, 10 cm/s, 20 cm/s,
10 deg/s, 20 deg/s, 30 deg/s, and

WEIGHT sensed by the wrist is 6 kg.

The numbers of cm are in cartesian coordinates, as shown in Figure III-7(a). The numbers of degrees are in the rotational

coordinates α β γ shown in Figure III-7(a).

For example, suppose that a robot is led through a task with no weight in its hand. Suppose that a weight is put in the robot's hand before it begins to execute the task again. Suppose that the weight affects only the first few movements of the task. The teacher verbally corrects these movements as the robot does the task. The robot stores productions, or three-tuples with (i) the step numbers from the stored sequence, (ii) the corrected actions, and (iii) the conditions, in the way shown in Figure III-9(a). Then, whenever the robot has to perform that task again, it senses the weight and selects the right actions from the productions, given the step number and the sensed weight. The teacher does not have to explicitly program conditional branches for each possible situation. He need only lead the task and correct the movements affected by each weight.

For a PSC to enable a robot to perform a task it must satisfy a condition of distinguishability. The information the robot uses to distinguish environmental situations, Cond in the three-tuple productions, must distinguish all those situations that are different for the task. The conditions, Cond, must contain enough information to tell which action is required in a particular situation of a task for producing the required movement. In the example task explained above, the robot needed to remember information about the weight on its wrist, and how it affected the movements caused by actions. Then the robot could select the correct action for producing a movement. What this means in general is that the robot must both (i) sense the prevailing conditions in the environment, and (ii) organize the sensory information in productions in a way that enables it to select actions for producing movements. The designer of the robot must ensure that

this condition of distinguishability is satisfied. The information required in the Conds of a PSC will be just the movement parameters that the ACS does not automatically compensate for.

III.4 GOAL-SEEKING (GS)

This section explains GS systems in detail. As explained at the beginning of this chapter, a GS system comprises (a) an ordered sequence of goals, and (b) a network of quintuples, or productions. It was also explained that a production has the form <$Cond_j$, C'_i, <Next $Cond_k$, prob(Next $Cond_k$ | j,i)>, <$Goal_\ell$>, Val> where $Cond_j$ is the robot conditions, C'_i is a motor command that will result in conditions $Cond_k$ with probability prob(Next $Cond_k$ | j,i), $Goal_\ell$ is a goal from the goal sequence, and Val is an estimate of how worthwhile it is to reach this production, in seeking the presently active goal. A GS system is depicted in Figure III-6, and in Figure III-10. Note that in the simplified productions of Figures III-10(a) - (c) and (e), no estimated probabilities are shown and each production has only one goal and one Next Cond. Figure III-10(d) gives examples of GS quintuples that are more complex than those in Figures III-10(a) - (c) and (e).

The network of quintuples in a GS system is also called a "Markov decision process" (MDP. Howard, 1960; Hartley et al, 1980). In fact the network is potentially several MDPs; one for each different goal in the goal sequence. In addition, each MDP changes in nature, as the probabilities change and as productions are added and changed. MDPs are discussed further in chapter VI. Chapter VI explains in detail how the Vals of quintuples may be calculated, enabling a robot to follow a path to its current goal.

During the leading of movements, motor commands would be put straight into productions in the GS system. However, the conditions (Cond) of the productions stored during leading would not have force information stored in them. The teacher is applying forces to the arm. So the forces sensed by the robot during leading will not be the same as those sensed during execution. For example, the top production in Figure III-10(b) would be stored when the movement +2 cm/s/s is led under the conditions 0 cm/s/s 3 cm, and then goal 2 is set by the teacher. Suppose that the robot has a 1 kg weight in its hand. If the teacher holds the hand of the robot, the robot would not sense 1 kg on its hand. The teacher takes the weight. No weight is stored in the condition during leading.

Not only is no force information stored during leading, but no estimated probability is stored in a production that is formed during leading. It is only when the productions are fully formed during execution that the probabilities can properly be estimated. Before that there are no weights in the productions.

During execution, the robot selects actions for reaching the active goal. Roughly speaking the robot makes a guess at which action will put it on the shortest path to its current goal. Initially only a guess can be made, since neither force information nor good probability estimates are in the led productions. Section 4.2 explains how such guesses can be made in a sensible way. As execution proceeds, the forces sensed would be put into the productions that had been stored during leading. If a Cond that has no force information stored in it---a "non-force Cond"---matches the non-force part of the robot's current conditions, then a copy of the production with that non-force Cond may be made. The current force conditions are put in the new copy

of the production, as shown in Figure III-10(c). The copy will not be made if a production with that Cond and C' already exists. In the Figure the robot executes +2 under conditions 0 cm/s/s 2 cm 1 kg, because the 0 cm/s/s and the 3 cm match the second top production shown. A new copy of the production is made, and the 1 kg condition put in, as shown by the top production in Figure III-10(c).

The original led production must be left intact, with its non-force Cond. Then another production can be formed, should <u>different</u> force conditions occur with those same non-force conditions. For example, suppose instead of 1 kg, 2 kg is sensed in the example of the last paragraph. Another copy would be made, with 2 kg instead of 1 kg in the Cond and Next Cond.

During leading a <u>number</u> of <C, Next non-force Cond, <Goal>, Val> four-tuples may need to be stored in a production with one non-force Cond. For example, consider the two productions

0 cm/s/s 1 kg 3 cm, +4, 4 cm/s/s 1 kg 6 cm, prob = 1, 2, Val_1

and 0 cm/s/s 5 kg 3 cm, -1, -1 cm/s/s 5 kg 2 cm, prob = 1, 1, Val_2.

During leading, only one production, with a non-force Cond of 0 cm/s/s 3 cm, would be stored for the leading of these two productions. The production would be

0 cm/s/s 3 cm, (+4, 4 cm/s/s 6 cm, 2), initial Val_1

(-1, -1 cm/s/s 2 cm, 1), initial Val_2.

During execution two different copies of this led production must be made, forming the two productions given above. So productions in a GS system may have this form

<non-force Cond, <C, Next non-force Cond, <Goal>, Val>>,

as well as this form

<Cond, C', <Next Cond, prob(Next Cond)>, <Goal>, Val>, which was

introduced at the beginning of the chapter. In Figure III-10(d) the top three productions would have been stored during execution, while the bottom production in the Figure would have been stored during leading.

Now, since several Cs may be stored in one production, as just explained, the robot may need to choose a C from a number of possible Cs. This will be necessary if its current conditions match a stored production which has in it more than one <C, Next non-force Cond, <Goal>, Val> four-tuple. The robot may select an incorrect C, because of the lack of force information in its Conds and Next Conds. Consider the example discussed in the last paragraph. In seeking a goal from non-force condition 0 cm/s/s 3 cm, it may be necessary for the robot to perform +4 when the sensed weight is 1 kg, but -1 when the sensed weight is 5 kg. For the different weights, the goal is reached by different paths. There might be an obstacle along the path for the lighter object, which the heavy object can be manoeuvred around only via a very long path. Suppose the current conditions are 0 cm/s/s 5 kg 3 cm. Suppose that the active goal is goal 3. Suppose the path from 0 cm/s/s 3 cm through the network to goal 3 via +4 is shorter than the path via -1. Since there are no weights in the stored production for condition 0 cm/s/s 3 cm, the robot may choose +4, regardless of the actual weight. However, the goal will not be reached via that path for +4, if the weight is 5 kg. There is an obstacle in the way. When +4 is performed the production 0 cm/s/s 5 kg 3 cm, +4, 2 cm/s/s 5 kg 6 cm might be stored, then another action performed, and so on. Next time the robot conditions are 0 cm/s/s 5 kg 3 cm, +4 may not be chosen. This is because the productions stored on the previous occasion show a longer path to goal 3 via +4, than the path via -1, if the weight is

5 kg. So -1 may be chosen.

If a "restart" button were available for a teacher to push, then the robot's learning of which action to choose might be speeded up. As soon as the robot started to execute something entirely wrong, the teacher could push the button. The robot would stop and then restart the task from the beginning. Next time the robot got to the choice point, the path it previously followed would have no goal as its end, and so wouldn't be followed. So for the example above, -1 would be chosen rather than +4. As will be explained in the appendix, a restart button may be useful for restarting the robot after a large movement error occurs. There may be no point in continuing after a large error. VC may be used to correct the error before the rest of the task is corrected.

When non-force productions are copied to make force productions, the goal code of the production should not always be copied. For example, when +4 was performed under conditions 0 cm/s/s 5 kg 3 cm, above, the production formed shouldn't have the goal "2" stored in it, even though the non-force production had "2" in it. The goal 2 is for a weight of 1 kg, not one of 5 kg. The robot will not know when to copy a goal code and when not to, since the weights are not stored during leading. Suppose the teacher has a single "special reward" button, which he pushes each time the robot reaches a goal during its first performance of the task. The special reward signal would indicate to the robot to copy the goal code into the force production being formed from the non-force production with the goal in it. The goal code would be deleted from the non-force production once copied. The teacher would be told to push the button when the robot first reached each correct goal.

As shown in Figure III-10(e), VC causes new productions to be formed. The production that was used by the robot is not changed or deleted. The goal-seeking of the robot must choose between the new and old productions next time their common Cond matches the robot conditions. The production with the highest Val will be selected. More will be said about this in section 4.2.

The steps a teacher would go through to teach a robot with a GS system are:

1 Put the robot into correction mode and teach verbal corrections. For example, the teacher might say "up" and lead the robot through an upward movement, and then say "wrist clockwise" while rotating the wrist clockwise, and so on until a repertoire of verbal corrections have been taught.

2 To start teaching a new task, type in a new task name, and then assign an empty goal list to the new name. A new task may also be set up by defining it in terms of other task names.

3 Lead the robot through the task. Put the robot into the leading mode. Press a "start recording" button when the robot is in the starting position and the task is set up to start. Lead the robot through the task, giving numbered rewards at each goal. A different number is used for each different goal. The goal number is combined with the task name when the goal is stored, so that goals from different tasks are not confused. Four-tuples of the form <non-force Cond, <C, Next non-force Cond, <Goal>>> are formed.

4 Start the robot executing, and make corrections. Push the "stop recording" button. Put the robot into the starting position, set up the start of the task, and put the robot into execution mode. The robot now performs the task as it has been learned. Verbal

corrections may be made by the teacher as the robot performs incorrect motor commands and when the robot gets into a set of conditions for which no production applies. The teacher may restart the robot part way through a task. Errors early in the task may need to be corrected before later ones. The teacher could restart the robot if it started doing the wrong part of a task. The first time the robot reaches each goal of the desired task, the teacher "rewards" the robot with the special reward button.

The teacher (a) uses his own ability at producing movements to teach the robot to produce movements, and (b) uses his own knowledge of the task to lead or verbally guide the robot to each of the goals. The robot selects motor commands for achieving the goal sequence. The network of productions may be retained from task to task, being continually added to as more productions are acquired. The robot learns to produce more and more movements.

Section 4.1 gives an example of a task for a GS system. Section 4.2 explains how one may resolve the conflict between seeking goals and finding out more information about paths to goals.

III.4.1 Example Task for a GS System

Consider the task described below as an example of one in which a robot must perform different movements, in different situations, in order to achieve the same goals.

Many robots are used for taking parts out of die-casting and injection moulding machines (Treer, 1979; Simons, 1980). The robot might have to put the parts onto a conveyor belt, as shown in Figure III-12. The robot might be doing the task all day. Various pieces of equipment and other parts may be in the way of the robot's

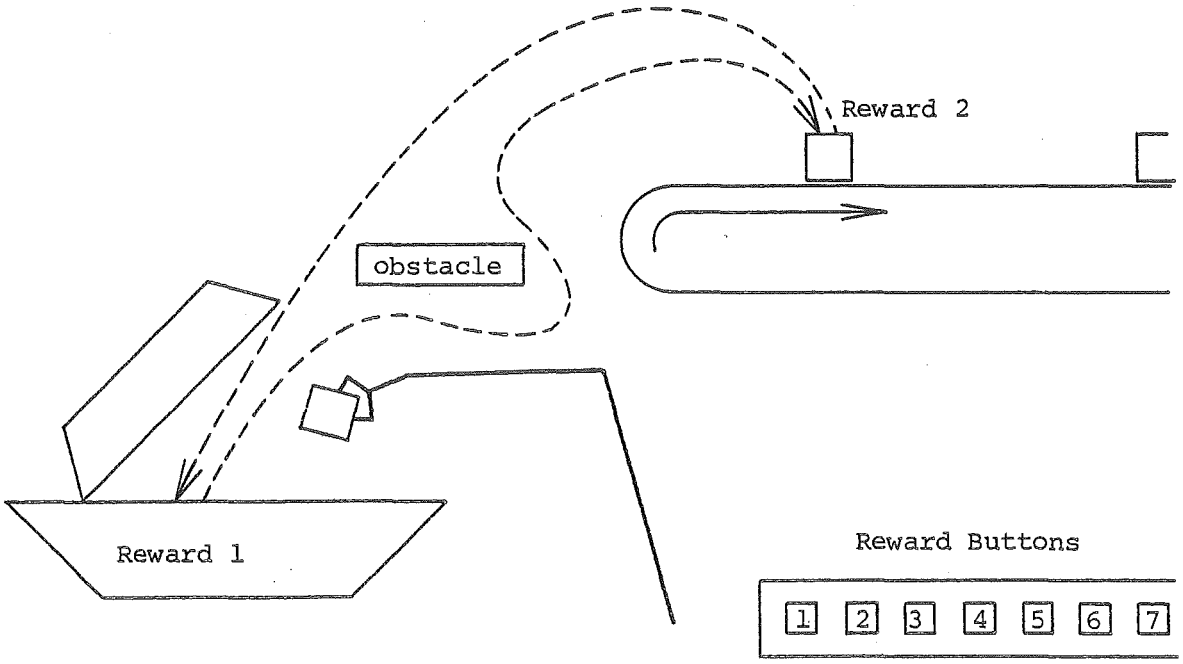


Figure III-12 The die-casting unloading task

The robot must put the parts from the die-casting machine onto a conveyor belt.

movements from time to time. Of course existing industrial robots are not obstructed in this way. It is a simple example of a task in which the robot must perform different movements in different situations.

Suppose a robot with a GS system has some sort of proximity sensor. The robot receives a signal whenever there is something near its arm. The information the robot gets tells it where the obstacle is in relation to its arm. Suppose the information from the proximity sensor is part of the conditions of the robot's productions.

The teacher sets up the die-casting transfer task by leading the robot through the task and pushing goal, or "reward", buttons. Two reward buttons set up two goals. The teacher first leads the robot to pick up a part from the die-casting machine. He pushes a button marked "Reward one", as indicated in Figure III-12. He then leads the robot to drop the part on the conveyor belt, and pushes a button marked "Reward two".

Led actions are put straight into productions with non-force Conds in them, as explained above, in section 4. For example if a movement M is led when the non-force robot conditions are $Cond_1$ then a production is stored with $Cond_1$ and C in it. C is the action which, in the popular implementation of the leading method, is stored in a sequence of actions when M is led, as shown in Figure III-2(a). A specific example of a production is given in the next paragraph.

Whenever the teacher pushes one of the reward buttons, the most recently learned or applied production is marked with that number of reward, as its Goal. For example, when the teacher leads open the robot's hand and pushes the Reward two button, the production given below might have Reward two stored with it. [The Cartesian and rotational coordinate systems, for the conditions in cm and degrees respectively, are shown in Figure III-7(a).]

Cond: HAND ACCELERATION = 0 cm/s/s, 0 cm/s/s, 0 cm/s/s,
0 deg/s/s, 0 deg/s/s, 0 deg/s/s;
WEIGHT = ? kg; HAND POSITION = 50 cm, 50 cm, 50 cm;
HAND ORIENTATION = -90 deg, 0 deg, 0 deg;
HAND VELOCITY = 1 cm/s, 1 cm/s, 1 cm/s,
0 deg/s, 0 deg/s, 0 deg/s;
OBSTACLE = directly below the hand;
HAND: Closed.

C: OPEN HAND

Next Cond: (this Cond would be stored here after the OPEN HAND action occurs,
HAND ACCELERATION = 0 cm/s/s, 0 cm/s/s, 0 cm/s/s,
0 deg/s/s, 0 deg/s/s, 0 deg/s/s;
WEIGHT = ? kg; HAND POSITION = 50 cm, 50 cm, 50 cm;
HAND ORIENTATION = -90 deg, 0 deg, 0 deg;
HAND VELOCITY = 1 cm/s, 1 cm/s, 1 cm/s,
0 deg/s, 0 deg/s, 0 deg/s;
OBSTACLE = directly below the hand;
HAND: Open.)

Goal: ("2" is stored here)

Val: (Val will be calculated once a goal is activated during execution.)

The obstacle below the hand is the conveyor belt. Accelerations are included in the conditions so that the robot is sensitive to the acceleration changes caused by actions. Later, during execution, copies of this production will be made, with weights and estimated probabilities being added. For example the weight in the Cond above may be 1 kg; the weight of an object. The weight in the Next Cond

would be 0 kg, since the object has been released.

The change in conditions from the Cond to the Next Cond given above, is just that the hand becomes open. During execution it will also be found that the weight on the wrist decreases by 1 kg, to zero. The teacher led the robot to drop the part onto the conveyor belt.

Now, whenever Reward two is operating, the robot tries to find a sequence of actions that cause transitions which get it to the Reward two condition above. It tries to find a sequence of actions that produces a sequence of condition changes, a path through the network of productions, which finally ends up at the Reward two production shown above. Then it can do the OPEN HAND action. It has reached the Reward two goal set up by the teacher. As will be explained in Chapter IV, once the value, Val, of each quintuple is known, the robot can follow the shortest path through its internal representation to the Goal that is operating, by selecting the active production with the highest Val. The active production is the one whose conditions, Cond, are the actual conditions the robot is in. The method of "Policy-iteration" has been proposed (Howard, 1960) for solving an MDP. An action that optimizes the reward gained by the robot is found for each set of conditions. Another method, called "leak-back", of solving MDPs in an MCLS is discussed in Chapter VI, and in MacDonald (1982a).

The teacher pushed the reward buttons in the sequence Reward one, then Reward two. This makes the robot seek Reward one, then Reward two, then Reward one, etc. That is, the robot searches for actions which enable it to pick up a part, and then transfer it to the conveyor belt, and then pick up another part, and so on.

There may be an obstacle sensed by the robot while it is moving its arm. There may be a production in the GS system which, in the presence

of such an obstacle, puts the robot on a path to its current goal. The robot may perform the action in that production. If there is not such a production in the GS system, the robot might stop, make a noise and flash a light. The attention of a human might be attracted. The human could use verbal corrections to move the robot around the obstacle. The robot would remember the actions in productions and use them for avoiding the obstacle later on. VC of the GS system is shown in Figure III-10(e). The human could also lead the robot around the obstacle. In either way a path to the goal may be completed.

The GS system a robot has must satisfy a condition of distinguishability. Just as the conditions of productions in the PSC must contain enough information to select the right action for producing a movement---see the last paragraph in section 3---the conditions in the GS system must contain enough information to select the right action for getting nearer the goal. The conditions in productions must also properly distinguish goals. Only if a task or subtask achievement can be represented by one of the productions could the task or subtask be set up as a goal for the robot. If the conditions in the productions do not enable the reaching of the true goal to be distinguished from not reaching it, then the goal could not be represented properly. The robot designer must ensure that a GS system satisfies the condition of distinguishability for the sorts of task the robot must perform.

Finally, there is a disadvantage to the example GS system described in this section. The goals would have to be set up each time the teacher wanted to get the robot to do a different task. It would be easier for the teacher if he had to teach and set up the task only once. For example, if he could say "Do die-casting transfer task",

having once, some time ago, led the die-casting transfer task, set up the reward goals and made any corrections required. At the start of this chapter, and at the start of this section, 4, it was suggested that a task name be given by the teacher so that the robot could remember the goal sequence for a task. Then the teacher would have to teach a task only once. Task goals could be represented in the way shown in Figure III-10(d).

In the end I want to be able to tell the robot what to do using natural language (see Andreae, 1977a, 1978, 1979a, 1979b; Kuiper, 1980a, 1980b; MacDonald & Andreae, 1981). I have been more concerned with the motor abilities of the robot here though.

III.4.2 Dual Control

Note that in seeking a goal, the robot must select actions in a way that compromises between (a) discovering the Cond changes, or "transitions", caused by actions, and (b) reaching a goal using the information it has already obtained about the transitions. That is, the robot must compromise between taking a known path to a goal and exploring other, lesser known paths. This is because, by exploring lesser known paths, a robot may find a path to the goal that is shorter than any others it knows.[6] The transition caused by an action when it is led may not be the same as when the action is performed. So the robot must sometimes perform an action to discover the transition it causes, rather than to achieve a goal most quickly. Only if the incorrect led transitions are corrected can one expect the robot to find the shortest paths to its goals.

Now recall from the very beginning of this chapter and from section 2 that opposing forces may affect the performance of an action.

Opposing forces will not disturb the effect of a led action because the led actions are derived from the actual movements. The teacher compensates for opposing forces during the leading of a movement. So "incorrect" transitions may be stored during leading.

For example, an action UP +10 might cause an acceleration change of +5 cm/s/s upward when there is a weight on the arm, but of +10 cm/s/s upward when the arm is unloaded. The teacher would have to lead a +10 cm/s/s upward movement in order to teach an UP +10 action. If the arm is being led, then the effect of an UP +10 action is +10 cm/s/s upward, whether the arm is loaded or not. The teacher takes the weight of the load. The UP +10 is stored in a non-force production, say P. A transition from P to a production whose acceleration condition is 10 cm/s/s greater than P's, will then be stored when the next action is led. When the UP +10 is performed, a copy of the non-force production would be made, but with a Next Cond whose acceleration condition is +5 cm/s/s more than P's. Only when all transitions are "correct" may the robot select the shortest path to every goal.

The trading-off between obtaining more information, or "exploring", and reaching goals using the information obtained so far, or "controlling", is called the problem of 'dual control' (Fel'dbaum, 1960; Cashin, 1970; Witten, 1976; Andreae, 1981). Usually the requirements of exploration and control conflict.

There are special ways of choosing actions, which give a good trade-off between exploration and control (Cashin, 1970). Cashin suggests that an action should be chosen with the probability that it is the best action. As well as an estimate of the value (Val) of a production, an estimate of the accuracy of the value estimate is stored. As the robot's experience accumulates the estimates of values

will become more accurate, as long as its environment is stationary. The transitions in the quintuples of a GS system are a list of two-tuples, <probability(Next Cond), Next Cond>. The more a robot performs an action, the more accurately can be estimated the probabilities in the two-tuples. Thus the action's value can be estimated more accurately.

Actions are chosen on the basis of both their values and the accuracies of these values. The apparent best action is more likely to be chosen if the accuracies are all high. If the accuracies are low then some other action has a greater chance of being chosen, than if the accuracies are high. To start with, the accuracies of actions' value estimates will be low, so apparent non-optimal actions will be chosen reasonably often. Thus the accuracies of all estimates will increase as experience accumulates. Cashin shows that such a scheme is a near-optimal solution to the dual control problem.

Note that an action should be selected from among both (a) the productions with force Conds that match the current robot conditions, and (b) the production with a non-force Cond that matches the current robot conditions. If there is an action in the non-force production that is not in a force production, then it should have a chance of being performed, in case its true value is highest.

If the environment is non-stationary then the "correct" transitions may be changing, so there must be a non-zero chance of selecting any available action in each situation. Then the internal representation may keep up with the changing environment characteristics.

The problem of a robot finding the best actions to do when it has imperfect information about the effects of its actions on the environment, is similar to the problem of a climber finding the top of

an unknown hill in mist or fog. If you simply keep climbing upwards you will eventually come to the top of something. However, it may not be the highest peak on the hill. By exploring the hill you may find a path to a higher peak.

The simplest form of the dual control problem is the classical "two-armed bandit" problem (Bellman, 1961; Cashin, 1970). The two-armed bandit problem is this. One has two slot machines. There is a fixed probability that each machine will pay out a dollar when its handle is pulled. The pay-out probability of one machine is known. The pay-out probability for the other machine is unknown. In another version of the two-armed bandit problem both pay-out probabilities are unknown (Witten, 1976). The object is either (a) to maximize the total number of dollars received over a finite number of handle pulls, or (b) to maximize the total discounted future dollars received over an infinite number of handle pulls. There are other slightly different maximizations that can be made (Witten, 1976). However, these two are the most interesting for this chapter. The finite length maximization in (a) is the sort that would be made by a robot doing a task for a finite length of time. The discounted infinite length maximization in (b) is the sort that would be made by a robot doing a task for a very long time, where there is a non-zero probability that the robot will stop performing the task before that time is up. The discounting factor that is applied to the expected future rewards may represent the probability that they will be received (Howard, 1960). Both maximizations involve a trade-off between (i) finding out the unknown probability or probabilities, by pulling the handle of the machine you know least about, and (ii) pulling the handle of the machine you think is best.

An optimal trade-off can be computed for the two-armed bandit problem (Bellman, 1961; Cashin, 1970; Witten, 1976). However, in general the method used is computationally impractical (Cashin, 1970; Witten, 1976; Gittens, 1979). The solution cannot be obtained analytically, and numerical approximations are too "unwieldly" for practical use (Kumar & Seidman, 1981). The solution becomes more difficult (a) when there are more than two choices of action, and (b) when the best action to perform depends on the state of the environment. Recently, a more practical method has been described which computes an optimal trade-off for the two-armed bandit problem (Gittens, 1979). However, the method is not optimal for more complex systems, such as MDPs. Cashin's method, mentioned above, provides a near-optimal solution for MDPs. As stated at the start of section 4, the quintuples of a GS system form a number of MDPs, one for each goal.

III.5 CONCLUSION

I have proposed two paths of improvement to the popular leading method, which is a natural, easy to use, widely employed method of teaching a robot a sequence of movements. If a teacher uses popular leading then he must do explicit programming, for editing motor commands and for forming conditional branches. The improvements would overcome the need for explicit programming by enabling a teacher to teach a robot using only his natural ability at leading and verbal correcting, and his natural knowledge of task goals.

On the first path, an on-line verbal correcting (VC) scheme is added to popular leading, enabling a teacher to teach a robot successive corrections to the actions it sends to its arm control system (ACS). The teacher could correct led actions without having to edit them via a keyboard device. Verbal corrections may correct for (a) incorrectly led movements, and (b) opposing forces that are not automatically counteracted by the ACS. Verbal corrections themselves are taught by the leading method. Successive VC will be stable and convergent under reasonable conditions.

Also on the first path, a production system of corrections (PSC) is added, enabling a robot to remember and use conditional corrections for opposing forces. The robot selects its own actions for achieving movements. To teach the robot to perform a sequence of movements under a variety of conditions the teacher need only lead the sequence, and verbally correct individual movements, rather than having to explicitly program conditional branches for different situations.

Path 1 improvements would be suitable for teaching a task comprising a sequence of movements.

On the second path of improvement, a goal-seeking (GS) system replaces the sequence of actions stored in popular leading and in the improvements of Path 1. This enables a robot to remember its experience of producing movements with actions, in a network of productions, or quintuples. It can select its own actions for achieving a sequence of goals. A teacher need only set up a goal or goals, and teach individual movements, rather than teaching all the various sequences that may be required for achieving the goal. Also on the second path, VC is added to GS. The teacher can teach the robot new productions as the robot seeks a goal, either when the robot stops, or when it makes "mistakes".

The two paths of improvement would increase the teachability and adaptability of led robots. VC should increase the teachability by enabling a teacher to correct motor commands using his own natural ability at verbally correcting humans. A PSC and a GS system should increase the teachability by enabling a teacher to teach conditional branches using his natural leading and verbal correcting abilities and his knowledge of the subgoals of the task he leads; a PSC for sequential tasks and a GS system for goal tasks. A PSC and a GS system should increase the adaptability of led robots by enabling a robot to select its own motor command, given the sensed robot conditions. In a PSC motor commands are selected for performing a sequence of movements. In a GS system motor commands are selected for achieving a sequence of goals.

NOTES FOR CHAPTER III

1. Some writers use the term "lead-through" to refer to the method of guiding a robot via some control device; for example, Morris (1982) and Simons (1980). Some writers use the phrase "manual guiding" (Grossman & Taylor, 1978), or "guiding" (Lozano-Perez, 1983), for the method of physically moving a robot through movements. I shall use the words "lead", "led", "leading", "lead through", et cetera, to refer to the method of physically moving a robot through movements. I shall call methods which use a control device "guiding" methods. Guiding methods were discussed in Chapter II.

2. Cunningham (1979) explains that a led motor command may be edited "on-line" by a teacher moving the robot to the point in the led sequence where the change is required, and then making an insertion or deletion via a keyboard. This makes editing easier than using just a keyboard. However, the method would work only for a robot that recorded positional motor commands, since the dynamic characteristics of the movements could not be so easily edited. As pointed out in Figure III-2 and in section 2, a robot may need to have accelerations as motor commands.

3. Hyman (1953) found that human reaction times were about 200 ms plus 200 ms per bit of information transmitted. Thus for the 36 corrections of Figure III-7, one could expect reaction times of about one second. Then there is the choice of "larger" or "smaller" before a correction, giving 108 possible combinations. The reaction time would be about 1.5 seconds. Crossman (1953) reports somewhat larger reaction times for card sorting tasks. Welford (1980b) states that many questions about reaction times remain unanswered. Now, the reaction time for a response to a stimulus from a set of possible stimuli and responses is considerably decreased if (a) the response for a stimulus is seen by humans as naturally compatible with the stimulus (Marteniuk, 1976; Sheridan & Ferrell, 1981); and (b) the human has had a lot of practice at responding to the stimuli (Marteniuk, 1976; Sheridan & Ferrell, 1981; Welford, 1980b). Verbal corrections are naturally compatible with movement errors in the sense that a verbal correction is just a verbal expression of the change in movement needed to correct a movement error. So after some practice it would be reasonable to expect a human to make verbal corrections with a delay of about one second. The time taken for a robot to recognise the words and select a correction should be a very small fraction of a second. The delay between an "incorrect" action being performed and an action correction being made should be about one second, if the teacher does not anticipate the incorrect action.

4. Even if a teacher does know the right sequences of actions for a task, he may not be able to lead them. The leading of a task whose movements are not automatically compensated by the ACS gives the robot the wrong internal representation. Is it possible to lead a sequence of movements that gives the right internal representation? Well, it may actually be <u>impossible</u> for a teacher to lead a robot through the right sequence of actions for a task. A task for an industrial robot might be to lift objects up onto a conveyor. The leading of the sequence of actions required for lifting a heavy object might take the

arm way above the conveyor. Actions for lifting higher are needed for compensating for the heavy object. The next actions in the sequence for the task would be actions to move the arm and object over the conveyor. The structure supporting the conveyor might prevent the teacher leading these next actions way above the conveyor.

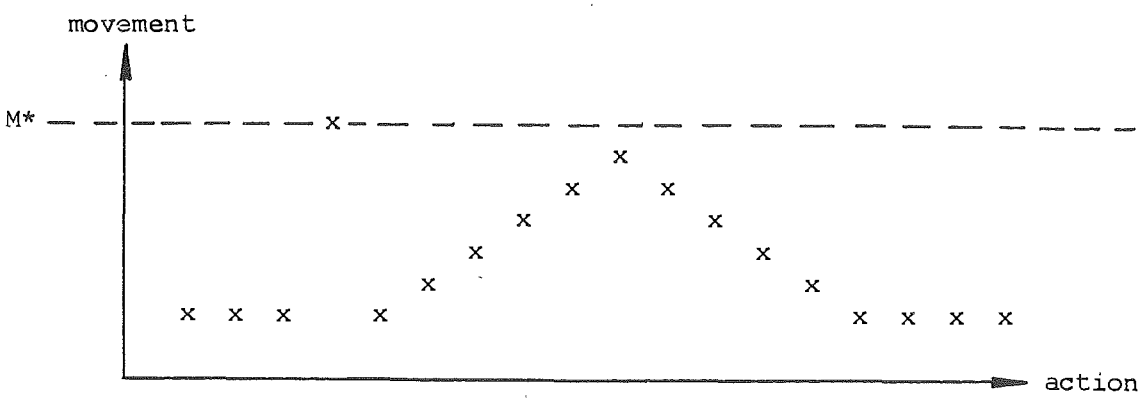
5. In order to explicitly program a robot to perform a task whose movements are not automatically compensated for by the ACS, a teacher must obtain an explicit representation of the compensations the robot must make, as well as an explicit representation of the task movements. Note also that guiding methods enable a teacher to teach a robot how to compensate for opposing forces, during the guiding of the task. During guiding, the teacher sends actions to the ACS, via a control device; for example a keyboard. He shows the robot the actions it must send to the ACS. The robot "takes the weight" during the guiding of the task.

6. As well as new transitions being formed, force information will be put into productions, during execution. However, this is not important for dual control because the force information itself cannot show up a new path through the network to a goal. Without force information in it a production is ambiguous with respect to force. This effectively provides for paths through that production which have any forces at all, as long as the other conditions are matched with the Cond. When a copy of that production is made, and force information put into the copy, it may be found that that particular production does or does not lead to the goal. However, no new path can be formed, unless a new transition is formed.

7. The words "smaller" and "larger" may be used by a teacher to form corrections that have not been led, as suggested in section 1. The words "smaller" may be used to give smaller and smaller corrections, down to the smallest possible.

8. There will be correcting situations that are worse than that shown in Figure III-14. For example, Figure III-13 shows a one-dimensional relationship between movements and actions in which there is a "false peak". The teacher would have to realise at some stage that his correcting on the shallow, false peak could not produce M*.

Figure III-13
A worse correcting situation than that shown in Figure III-14



APPENDIX FOR CHAPTER III

III.A.1 Verbal Correcting is Stable and Convergent under Reasonable Conditions

To begin with I will establish the stability and convergence of successive verbal corrections applied to a single movement under a particular set of conditions. Then I will extend the single movement case to the successive correction of a sequence of movements.

It is assumed that any particular action always produces the same particular movement under a particular set of conditions. If this were not the case then an action that produced the desired movement under the required conditons might also produce other movements under those same conditions. For example, one could not expect a teacher to successfully verbally control or correct a robot arm that was being subject to very complex force interactions with, for example, a human being. The teacher would certainly be unable to correct the robot's movements if he could not know what forces the human would apply to the robot's arm as a result of one of the teacher's corrections. It is also assumed (a) that there is an action that produces the desired movement, (b) that there exists a sequence of leadable correction actions that add up to the correction required and (c) that the teacher knows the movements he wants. Leadable corrections are discussed in section A.2.

I shall explain that a teacher will be able to satisfy Dvoretzky's conditions (Dvoretzky, 1956; Wilde, 1964) for the convergence of a stochastic approximation scheme. There will be some very complex situations in which correcting will be very difficult, but not impossible, for a teacher. The proof will be illustrated with an example verbal correcting strategy. A teacher might be instructed to follow this strategy. The strategy comprises three rules to follow:

R1 Don't give up correcting just because the movement errors on successive corrections do not seem to be decreasing. Be assured that they will eventually decrease. Only give up if you have exhausted the entire range of actions.

R2 If a correction causes the movement error to be overcorrected in any dimension then make your next correction smaller. The more overshooting occurs, the more you should reduce your corrections. You must observe movement errors in one particular set of dimensions, or coordinates. It doesn't matter which coordinates you use, as long as you are consistent.

R3 The best way to correct a movement is to try and overcorrect in each dimension, and then correct, according to R2, back the other way.

In addition Figure III-8 gives a strategy which should make VC easier.

(1) represents a teacher's nth verbal correction, V_n, to a robot movement. M_n is the movement achieved after the n-1th correction. M_{n+1} is the movement achieved after the nth verbal correction.

(1) $M_{n+1} = M_n + V_n$.

Movements and verbal corrections are acceleration changes, as explained in section 2.

Now a teacher may select a verbal correction partly on the basis of the results of previous corrections. So we may write

(2) $M_{n+1} = T_n (M_1, \ldots, M_n)$.

There may be random fluctuations, or noise, N_n, added to (2). So,

(3) $M_{n+1} = T_n + N_n$

It will be explained why these random fluctuations are important, in the discussion below of the correction of a sequence of movements.

Dvoretzky (1956) has shown that successive approximation schemes like (3), called stochastic approximation schemes, will converge on M*, the desired movement, if certain conditions are satisfied. A scheme is stable and converges in the sense that both

(4) $\lim_{n\to\infty} E[\,|M_n - M^*|^2] = 0$

and

(5) $P[\lim_{n\to\infty} M_n = M^*] = 1,$

where P means "probability" and E means "expected value". The first condition that must be satisfied is

(6) $|T_n(r_1,\ldots,r_n) - M^*| \leq \max[a_n, (1 + b_n)|r_n - M^*| - c_n]$

for all real $r_1, \ldots, r_n$. The possible values for r_i, i = 1 to N, are just the possible movements---the possible M_i's---any of which might be produced during a teacher's successive verbal correcting. In (6), a_n, b_n and c_n are non-negative real numbers satisfying

(7) $\lim_{n\to\infty} a_n = 0,$

(8) $\sum_{n=1}^{n=\infty} b_n < \infty,$

and

(9) $\sum_{n=1}^{n=\infty} c_n = \infty.$

These three conditions must also be satisfied,

(10) $E[\,M_1^2\,] < \infty,$

$$(11) \qquad \sum_{n=1}^{n=\infty} E\,[\,N_n^2\,] < \infty,$$

and

$$(12) \qquad \sum_{n=1}^{n=\infty} E\,[\,N_n\,] \text{ is bounded and convergent}$$

Now (4) through (12) apply to stochastic approximation in one dimension. For multiple dimension stochastic approximation (for example, the six dimensions of translational and angular acceleration for a robot arm) each absolute value in (4) through (12) is replaced by a vector norm, and each square in (4) through (12) is replaced by the square of a vector norm (Wilde, 1964; Dvoretzsky, 1956). A vector norm is a measure of distance.

(10) is satisfied as long as the first, uncorrected movement is finite. (11) means that random fluctuations must vanish in the long term. (12) means that any bias in the random fluctuations must vanish in the long term. Random fluctuations are discussed shortly.

Briefly: (7) ensures that overshoot in each dimension decreases; (8) ensures that in each dimension any tendency for M_n to go further away from M*, vanishes; and (9) ensures that correcting power is infinite, so correcting does not stop short of M*. Briefly, the example strategy satisfies these three conditions for three reasons. Firstly, R2 ensures that overshooting will decrease in the long term, because the teacher will keep on reducing his corrections until it does (also see note 7). Secondly, there will be a finite range of movements possible. So the possible increase in movement error is finite. Since the possible total error is finite, successive increases in error must eventually vanish. Thirdly, R1 ensures that the teacher does not stop correcting until the error is corrected.

Now (6) is true as long as the left hand side is not greater than at least one of the two right hand terms. For each dimension of movement space, a correction may (i) cause overshoot in the movement error in that dimension, (ii) cause undershoot in the movement error in that dimension, (iii) cause the error in that dimension to be reduced to zero, or (iv) cause an error to be produced in that dimension where previously the error was zero. (iv) may be treated as a special case of (i). I will argue that a teacher's verbal correcting, represented by (3), ensures

(a) that the left hand side of (6) is not greater than the first term on the right hand side of (6), whenever correcting overshoots M* in any dimension. That is, overshooting tends to decrease in the long term;

and (b) that the left hand side of (6) is not greater than the second term on the right hand side of (6), whenever correcting undershoots M* in any dimension. That is, increases in error vanish and correcting power is infinite.

Then (6) is satisfied.

Note that overshooting in one dimension may occur at the same time as undershooting in another dimension occurs. In this case I will argue that the left hand side of (6) is not greater than both terms on the right hand of (6).

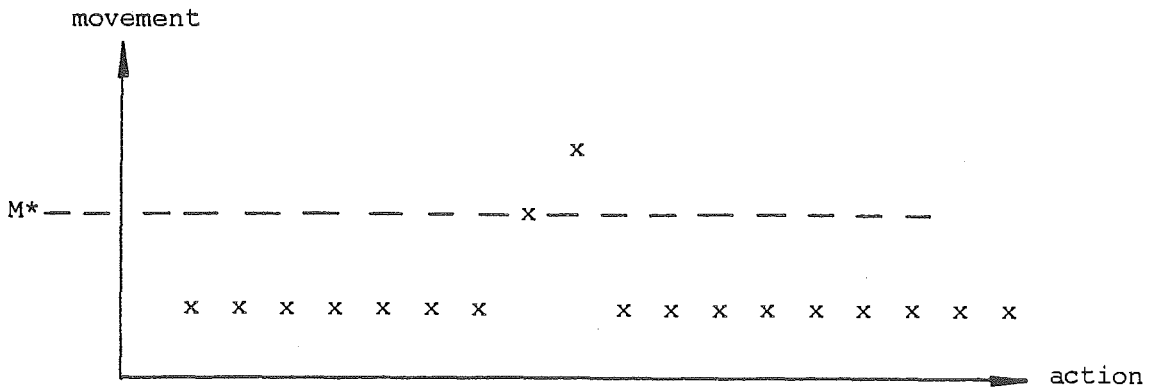
For (a) to be true, the amount of overshoot in movement corrections must decrease in the long term, so that (7) is satisfied. If overshoot increases in any dimension, a teacher will naturally reduce the size of his corrections. For example, he might use "creep up" to correct after a downward overshoot results from "left" (see also note 7). In the example strategy, the teacher reduces the size of his corrections every time overcorrection occurs in any dimension. If overcorrection continues he makes more and more reduction in his corrections. Since I have assumed that there is only one movement produced by a particular action under a particular set of conditions, any finite change in action causes a finite change in the movement. So there must be a size of correction that does not cause overcorrecting.

For (b) to be true two conditions must be met. Any increases in error, that is, any moving away from M* in any dimension, must vanish as correcting goes on. This is expressed by (8). Also, decreases in error must not vanish. The second condition will be satisfied if the teacher goes on correcting until the error is reduced to a level acceptable to him. R3 and R1 in the example strategy encourage a teacher to go on correcting, even when the error is not decreasing. The first condition will be true because the movement range in each dimension is finite, and therefore the possible error is finite. Thus, successive error increases must eventually vanish. The sum of an infinite number of error increases must be finite, so (8) is satisfied in (6).

Only rarely during VC will the error actually increase to the maximum error possible. Often the teacher, having just increased the error, will be able to change his verbal correcting so that the error starts to decrease again. For example, there will be unusual situations in which a correction caused by "up" will make the arm go further down, or perhaps further to the left. Humans can cope with some situations of this nature. For example, to turn a bicycle quickly to the left one must first turn the handle bars to the right, in order to lean the bicycle over to the left.

So both (a) and (b) are true. Therefore the teacher's verbal correcting of a single movement is stable and convergent, under the assumptions made in the second paragraph of this section, as long as the conditions (11) and (12), on random fluctuations, are satisfied. I assume that random fluctuations caused by hysteresis, friction, stiction, etc will not be large enough to be significant.

Figure III-14 Example one-dimensional graph of movements versus actions



There will be some particularly difficult correcting situations. For example, imagine that the graph in Figure III-14 shows the relationship between movements and actions in one of the dimensions of the arm. Using VC the teacher may have to search the entire range of actions, in order to find the one required, since a wrong action gives no hint as to what sort of correction is required (see also note 8). It would be very unusual for a robot arm to perform exactly the same movement for every possible action except those in a very small range.

The verbal correction of a sequence of movements will now be discussed. Consider the kth movement in a sequence of movements. If any of the 1st through k-1th movements have not been fully corrected, then the robot arm's position, velocity and acceleration may not be what the teacher wants them to be at the start of the kth movement. Therefore (a) the error in the kth acceleration change may not be what it would have been if there were no errors in the 1st through k-1th movements (for example disturbing forces may be different or may have a different effect on the arm if the arm has a different position, velocity or acceleration), and (b) the actual acceleration may be wrong even if the kth acceleration change isn't.

The interference to the kth correction of error from the 1st through k-1th movements may be represented in the stochastic approximation scheme as the random fluctuations, N_n. M_{n+1} will depend on (a) M_n, (b) the teacher's correction, V_n, and (c) the errors in previous movements, say N_n. The stochastic approximation scheme can cope with any fluctuations that obey conditions (11) and (12). As long as the fluctuations eventually vanish, the kth movement's corrections will be stable and will converge, by the argument above.

Now, if I can assume that the 1st movement is not interfered with by previous movements, then the first movement will converge to M*, the desired first movement, by the single movement convergence argument given above, under reasonable conditions.

However, the robot may have to perform a sequence of movements which goes on continuously, the end merging into the start without the arm stopping. Then the 1st movement could be influenced by the errors in the movements of the previous cycle. However, the teacher can always provide a lead-in to such a sequence, from a stationary start. He can always set up the sequence so the first movement converges.

Having corrected the sequence of movements, the teacher may have to remove the lead-in. If the robot had one of the internal representations used on improvement path 1 then the teacher would have to program a simple branch in the sequence so that the proper cycle was performed. The branch would go from the end of the sequence to the first movement after the lead-in. I wouldn't expect a lead-in to be required for many tasks. The ability to have a lead-in taught and later removed, might be designed into the robot to make teaching easier. A robot with a GS system would skip a lead-in sequence if that sequence wasn't needed for achieving its goal or goals.

Since the 1st movement converges, therefore the 2nd movement will converge, because the interference to it from the 1st movement will vanish as the 1st movement converges. So, therefore, will the 3rd movement converge, and so on. If all the 1st through k-1th movements converge then the interference to the kth movement will vanish, and the kth movement will converge. This is not to say that corrections to the kth movement will be of no use until the 1st through k-1th movements have converged, but only that the kth movement may not finally converge until the 1st through k-1th movements have converged.

In particularly difficult correcting situations a teacher may want to begin by correcting only the first few movements. He would go on to correct later movements only when the first movements had converged to the required values. He might want to repeatedly restart the robot after it has performed only part of the task.

III.A.2 Corrections that Can and Cannot be Made

As explained in section 2.3, a correction can be made if there exist a finite number of leadable corrections which add up to that correction. That is, a correction C'-C can be made if and only if there exist a finite number N of leadable movements, say M_i, i=1 to N, such that

$$(13) \quad \sum_{1}^{N} C_i = C'-C$$

where C_i is the action stored when M_i is led.

One might say that a movement M is strongly correctable if for every leadable movement, M_{led}, there exist M_i, i=1 to N, where N is finite, such that

$$(14) \quad \sum_{1}^{N} C_i + C_{led} = C'$$

where C_{led} is the action stored when M_{led} is led, and C' is the action that produces M.

A movement M might be called weakly correctable if there exists at least one such M_{led}. Then putting $C_{N+1} = C_{led}$ there must exist C_i, i=1 to N+1, such that

$$(15)\quad \sum_{1}^{N+1} C_i = C'$$

where C_i is the action stored when the movement M_i is led.

A movement M might be called non-correctable if there is no such M_{led}. That is, there are no such C_i, i=1 to N+1, satisfying (15). A non-correctable movement cannot be produced using VC.

A task is strongly correctable if and only if all its movements are strongly correctable. A task is weakly correctable if and only if at least one of its movements is weakly correctable, and none of its movements are non-correctable. A task is non-correctable if and only if at least one of its movements is non-correctable. A non-correctable task cannot be taught using VC.

A strongly correctable task can be produced by VC regardless of the movements actually led by the teacher. A weakly correctable task that is not a strongly correctable task can be produced by VC only from some led sequences.

III.A.3 "VC" Simulation

Figure III-15 shows two runs with a very simple VC simulation. The heading of the Figure explains that the keys to push are U for "up", D for "down", C for "creep", Z for "zoom", S for "smaller", L for "larger" and a space to do no correction. For example, the teacher could push the keys L, Z, U to produce a correction "Larger zoom up". The heading also explains that the simulated trajectory of three "movements"---just three integers in the simulation---starts at 100 200 300. There is only one dimension. The starting sizes of correction are also given. Finally given in the heading are the date, and the time elapsed since the simulation started running.

After the heading the two simulation runs are given. First the simulation step number and elapsed time are given. Then on the next line the three "movements" are given, each preceded by a correction if one was given. Run one shows how a large change in the trajectory, requiring large changes to the corrections, can be made in 15 steps and about 2 minutes. Run two also shows a large trajectory correction.

Figure III-16 gives the BASIC simulation program used to produce the results in Figure III-15.

III.A.4 Advanced PSC

The advanced PSC has the internal representation shown in Figure III-17. Essentially, this PSC is a production system of movements plus a production system of actions. I shall call it a PSC-M&A. Movement productions have the form

<step, Cond, movement>

Figure III-15 Verbal Correcting simulation runs

```
Verbal Correcting                                    20 April  1983

   B.A.MacDonald, University of Canterbury, NZ.  29 Jan. 1983.

Commands are U<p> or D<own>, and may be preceded by C<reep> or Z<oom>.
  <SPACE> skips.

L<arger> or S<maller> before a command scales all corrections in that
direction by a factor of  2  or  .5 .

Start at 100 200 300 for three values in sequence.

Sizes of correction:  1  5  20  for Up, and  -2 -4 -15  for Down

RUN ONE

04/20/83 00:00:07


 1  00:07
Larger Zoom Up  140     Larger Zoom Up  280     Larger Zoom Up  460
 2  00:37
Larger Zoom Up  460     Zoom Up  600     Creep Up  476
 3  00:50
Zoom Up  780      600     Smaller Up  516
 4  01:02
Larger Zoom Down  750      600     Down  508
 5  01:11
Larger Zoom Down  690      600     Creep Down  500
 6  01:18
Creep Up  698      600      500
 7  01:27
Creep Up  706      600      500
 8  01:32
Smaller Creep Down  702      600      500
 9  01:40
Creep Down  698      600      500
 10  01:45
Creep Up  706      600      500
 11  01:50
Smaller Creep Down  704      600      500
 12  01:55
Creep Down  702      600      500
 13  01:59
Down  698      600      500
 14  02:03
Smaller Creep Up  702      600      500
 15  02:09
Creep Down  700      600      500
```

Figure III-15 (continued)

RUN TWO

04/20/83 00:00:07

```
 1  00:07
Zoom Up  120    Larger Zoom Up  240    Larger Zoom Up  380
 2  00:34
Larger Zoom Up  280    Larger Zoom Up  560    Larger Zoom Up  1020
 3  00:44
Zoom Up  920    Up  720    Down  1016
 4  00:52
Creep Up  952    Smaller Zoom Up  1040    Larger Down  1008
 5  01:06
Zoom Up  1272    Smaller Larger Down  1024    Down  992
 6  01:26
Larger Zoom Down  1152    Creep Down  1008    Smaller Creep Up  1000
 7  01:38
Zoom Down  1032    Down  976    1000
 8  01:44
Down  1000    Up  1016    1000
 9  01:52
 1000    Creep Down  1000    1000
 10  02:05
```

Figure III-16 BASIC Verbal Correcting simulation

Results are in Figure III-15

```
1 CMD"CLOCK":CMD"DATE 04/20/83":CMD"TIME 00:00:00"
2 REM AMMENDED FOR LPRINTING 20 APRIL 1983
10 CLS:PRINT@0,"Verbal Correcting.  B.A.MacDonald, University
        of Canterbury, NZ.  29 Jan. 1983."
11 LPRINT"Verbal Correcting":LPRINT"   B.A.MacDonald, University of
   Canterbury, NZ.  29 Jan. 1983."
20 PRINT@64*3,"Commands are U<p> or D<own>, and may be preceded
            by C<reep> or Z<oom>.  <SPACE> skips."
21 LPRINT"Commands are U<p> or D<own>, and may be preceded by C<reep>
   or Z<oom>.":LPRINT"  <SPACE> skips."
25 FF=2
30 PRINT@64*5,"L<arger> or S<maller> before a command scales all
   corrections in that direction by a factor of "FF" or "1/FF"."
31 LPRINT"L<arger> or S<maller> before a command scales all corrections
   in that":LPRINT"direction by a factor of "FF" or "1/FF"."
35 PRINT@64*7,"Start at 100 200 300 for three values in sequence."
36 LPRINT"Start at 100 200 300 for three values in sequence."
40 REMINPUT"Put in size for Creep,Normal,Zoom Up and Creep,Normal,Zoom
   Down";C(0),N(0),Z(0),C(1),N(1),Z(1):X(0)=100:X(1)=200:X(2)=300
42 C(0)=1:C(1)=-2:N(0)=5:N(1)=-4:Z(0)=20:Z(1)=-15:X(0)=100:X(1)=200:
   X(2)=300
43 PRINT"Sizes of correction: "C(0);N(0);Z(0)" for Up, and  "
   C(1);N(1);Z(1)" for Down";
44 NN=0
45 FORK=0TO2:GOSUB1000:NEXT
46 LPRINT"Sizes of correction: "C(0);N(0);Z(0)" for Up, and  "
   C(1);N(1);Z(1)" for Down":LPRINTTIME$:LPRINT:LPRINT
47 NN=NN+1:PRINT@64*15,NN;:LPRINTNN;" "RIGHT$(TIME$,5):FORK=0TO2
50 FORJJ=0TO300:NEXT:PRINT@64*12,"
                                ";
51 PRINT@64*12+K*18,"";:GOSUB2000:IFQ$=" "THEND=0:GOTO100
60 IFQ$="L"THENPRINT"Larger  ";:LPRINT"Larger ";:F=FF:GOSUB200
   ELSEIFQ$="S"THENPRINT"Smaller  ";:LPRINT"Smaller ";:F=1/FF:GOSUB200
   ELSEF=0:GOTO75
70 GOSUB2000
75 IFQ$="C"THENPRINT"Creep   ";:LPRINT"Creep ";ELSEIFQ$="Z"
   THENPRINT"Zoom    ";:LPRINT"Zoom ";ELSEA$="":GOTO80
77 A$=Q$:GOSUB2000
80 IFQ$="D"THENPRINT"Down     ";:LPRINT"Down ";:I=1
   ELSEIFQ$="U"THENI=0:PRINT"Up       ";:LPRINT"Up ";ELSEGOTO50
85 GOSUB300
90 IFA$="C"THEND=C(I)ELSEIFA$="Z"THEND=Z(I)ELSED=N(I)
100 X(K)=X(K)+D:GOSUB1000:LPRINTX(K)"   ";:NEXT:LPRINT:GOTO47
200 RETURN:IFF=0THENRETURN
210 FORJ=0TO1:C(J)=C(J)*F:N(J)=N(J)*F:Z(J)=Z(J)*F:NEXT:RETURN
300 IFF=0THENRETURNELSEC(I)=C(I)*F:N(I)=N(I)*F:Z(I)=Z(I)*F:RETURN
1000 PRINT@64*10+K*18,X(K)"  "
1010 IFK<>2THENPRINT@64*11+(K+1)*18+2,"*";:RETURN
     ELSEPRINT@64*11+2,"*";:RETURN
2000 Q$=""
2010 Q$=INKEY$:IFQ$=""THEN2010
2020 RETURN
```

Figure III-17 Advanced PSC, PSC-Movements and Actions (PSC-M&A) The figure shows how:

(i) the motor command +5 is selected, as a result of the movement 2 cm/s/s being selected on step 1, and the robot conditions being 1 kg 2 cm;

(ii) a verbal correction of +2 is made, the teacher having said "up";

(iii) the robot performs +7, causing the movement 3 cm/s/s to be produced;

(iv) the movement in the movement production is changed from 2 cm/s/s to 3 cm/s/s. the teacher has "corrected" the movement; and

(v) the action production "3 cm/s/s, 1 kg 2 cm, +7" is formed. The teacher has shown the robot how to perform the movement 3 cm/s/s when the robot conditions are 1 kg 2 cm.

If there was neither a movement production nor an action production for a particular set of conditions then the motor command in the stored sequence would be sent to the ACS.

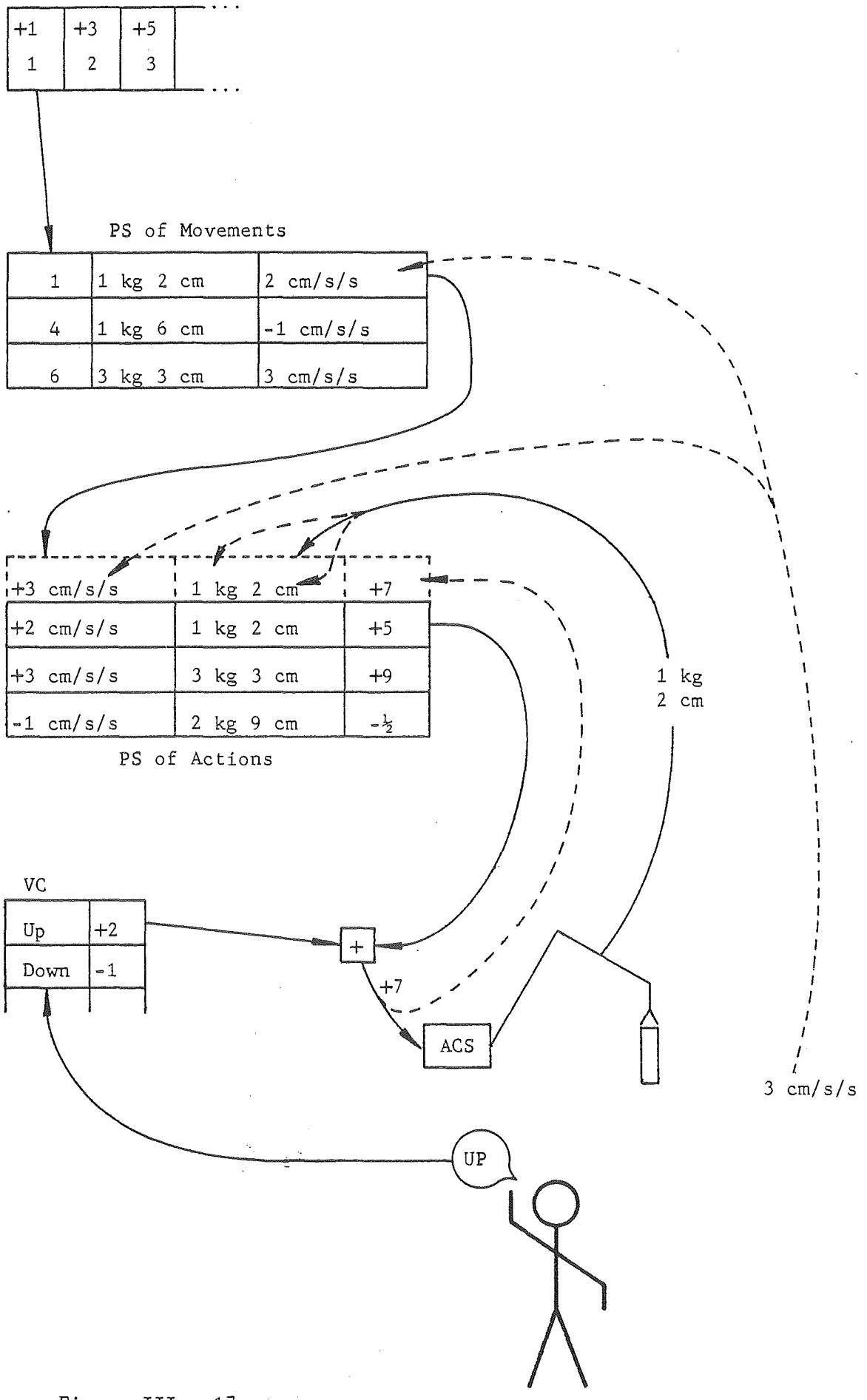


Figure III - 17 (caption opposite)

while action productions have the form

<movement, Cond, action>.

The PSC-M&A enables a teacher to correct the robot in the same way as one with a PSC. The only difference is that an action production formed on one step of the task can be used by the robot on another step of the task.

An added benefit is that the robot can keep its productions from one task to the next. As the number of productions increases the robot will know how to produce more and more movements. However, a teacher could not be sure that previous experience would provide actions for a new task; at least not for robots of the near future. So some teacher oriented method, for example VC, will be required for teaching action corrections.

As well as storing productions when VC occurs, productions should be stored when the robot performs "wrong" movements. For example, suppose the movement +1 cm/s/s upwards has been led, and hence the action UP +1 stored in the action sequence. When UP +1 is performed some force might oppose the arm, resulting in an actual movement of +½ cm/s/s upwards. The robot should store a production comprising the movement +½ cm/s/s, the performed action (UP +1) and the prevailing conditions. That production would not enable the robot to produce a +1 cm/s/s movement, but it would enable the robot to produce a +½ cm/s/s movement under those conditions. The robot's experience accumulates even without the teacher's VC.

Consider Figure III-17. Suppose the movement 4 cm/s/s upward, must be produced under the conditions 1 kg 2 cm on step 1. Suppose that the movement 2 cm/s/s is given by the PSC-M&A for step 1 and conditions 1 kg 2 cm. Suppose that the robot has a production giving an action UP +5 under those conditions when the movement required is 2 cm/s/s. The robot performs the action UP +5. Suppose that the teacher causes a verbal correction of UP +2 to be made to the performance of UP +5. The movement produced might actually be 3 cm/s/s upwards. Then an action production with the movement +3 cm/s/s on the left hand side, the conditions 2 cm 1 kg and the action UP +7 should be stored, as shown in Figure III-17, if it isn't already in the PSC. In addition, the movement production actually used by the robot will have its movement changed from +2 cm/s/s to +3 cm/s/s, as shown in Figure III-17.

Thus a PSC-M&A represents corrections to movements, and corrections to actions, separately.

CHAPTER IV

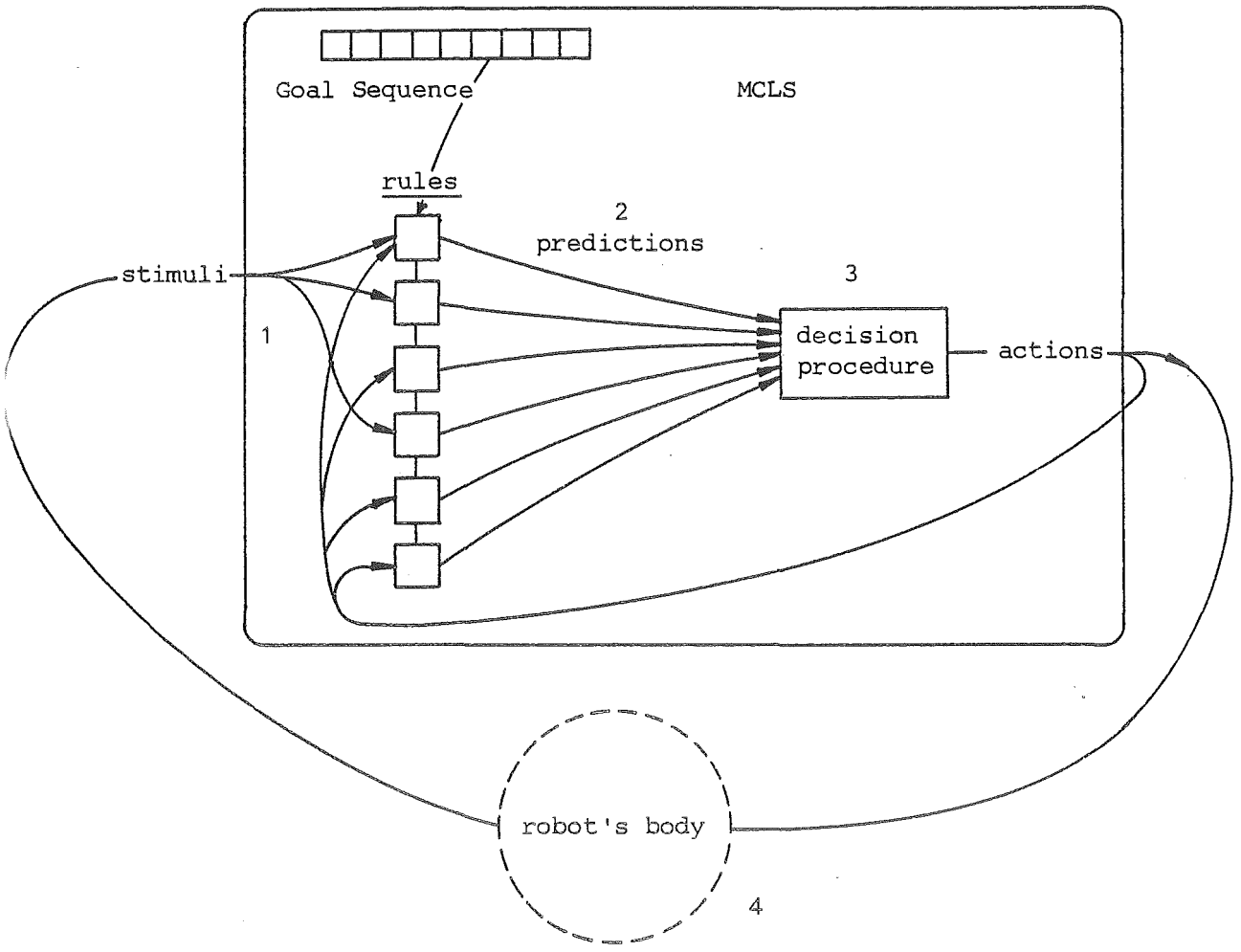
LEADING WITH AN MCLS

Chapter III described two paths of improvement to the popular leading method. The basis of path 2, a goal-seeking (GS) system, enables a robot to select its own motor commands, or actions, for achieving goals that have been set by a teacher. However, a GS system does not enable a robot to have an internal representation with sequences of movements or sequences of actions in it. For example, it would be no good a teacher leading a robot with a GS system through a painting sequence and rewarding it at the end. The robot would try to get to that end position along any path at all, not just the led path.

On the other hand, improvement path 1 of chapter III provides an internal representation with sequences in it, but no goals.

A multiple context learning system (MCLS) can provide an internal representation with both goals and sequences in it, as explained in section 1. An MCLS comprises a number of context systems, or "rules", a decision procedure, and a sequence of goals, as shown in Figure IV-1. Every MCLS in this thesis has only one goal in its goal "sequence". A rule is similar to a GS system, but it has contexts instead of Conds. A context is a set of recent actions and stimuli, whereas a Cond is just the set comprising the most recent stimulus of each type. For example, "UP +3, UP 4 cm/s/s, UP +2, UP 5 cm/s/s" is a context with two actions, UP +3 and UP +2, and two stimuli, UP 4 cm/s/s and UP 5 cm/s/s, in it. This context might have been formed when a robot performed UP +3, resulting in an arm acceleration of UP 4 cm/s/s, and then performed UP +2, increasing the arm acceleration to UP 5 cm/s/s. More

Figure IV-1 In an MCLS, the decision procedure selects actions to perform from the combined predictions of the rules. Each rule predicts actions for reaching the active goal. The operation step numbers 1 through 4 will be referred to in section 2.1.



precisely, the basic form of a rule comprises a list of quintuples, <Context, C', <Next context, probability (Next context)>, <Goal>, Val>. Section 2 explains that various reduced forms of the basic rule may be used in an MCLS. The decision procedure of an MCLS selects actions, to send to the robot's body, from the combined actions proposed by all of the MCLS's rules. Each rule of an MCLS is different in that the form of the <Context, C'> pairs is different in each rule. For example, one rule might have contexts which are the arm position and the arm velocity, while another has contexts which are the arm acceleration. Section 1 shows (i) how contexts containing recent stimuli and actions enable sequences to be taught to a robot, and (ii) how both sequences and goals can be taught to a robot when it has an MCLS with more than one rule.

In principle there is in fact no sequence of motor commands that an MCLS cannot produce, as shown in chapter VIII and in MacDonald & Andreae (1981). The practical abilities of MCLSs are discussed in several places (Andreae, 1977a; 1973-1983; Andreae & Andreae, 1978; Andreae & Andreae, 1979; Andreae & Cleary, 1976). However, not enough is known about MCLSs for me to make such definite statements about them as I have been able to make in chapter III about verbal correcting (VC), production systems of correction (PSCs) and GS systems.

It may be particularly difficult for humans to program very advanced "intelligent" robots to do the complex tasks that they will be capable of in the distant future. The leading method may be a useful teaching method for such robots, as outlined below and in the appendix. I would expect these advanced robots to learn also (a) by being told how to do a task, and (b) by watching someone else do a task. However, being told and watching may not be enough. Firstly, humans would find it

difficult to precisely describe complex tasks, such as hitting a tennis ball with a tennis racquet and riding a bicycle, in either high level programming languages or natural languages. Secondly, advanced future robots may find it difficult to learn parts of tasks involving certain intricate movements, such as hitting a tennis ball and riding a bicycle, by watching and being told. Humans find it difficult to acquire these tasks by watching and being told.

Leading a human through movements can be a useful as an initial teaching method (Sheridan & Ferrell, 1981). Leading provides a natural way of showing a robot what movements "feel like", as well as giving actions. Combined with the abilities of communicating in natural language and learning by watching, the leading method may well be used for teaching advanced robots of the distant future. As explained above, neither a GS system nor a PSC enable both goals and sequences to be taught. An MCLS may provide a realization of the leading method that is suitable for advanced robots' tasks, as well as for less complicated tasks that require an internal representation with goals and sequences. Some aspects of such an MCLS are given in the appendix. This section of the appendix is necessarily somewhat speculative. I explained in chapter II that existing robots are neither very teachable, nor very adaptable. We know very few hard facts about what a "very advanced intelligent" robot will be like.

The way an MCLS can provide an internal representation with both goals and sequences is described in section 1. Some simple examples of MCLSs are given. It is explained that an MCLS can implement VC by having a rule that stores verbal correction two-tuples as <Context, C'> pairs. The basic characteristics and terminology of MCLSs are explained in section 2. Section 3 briefly mentions previous work on

MCLSs and leading.

The remaining chapters of the thesis are concerned with establishing certain important abilities of MCLSs. I demonstrate leading with a simple MCLS and a simple, single-jointed arm, in chapter V. Chapter VI shows that optimal goal-seeking can occur in an MCLS. Chapter VII establishes that a robot can learn to perform reflex eye movements in non-reflex situations. As will be explained in section 1.1 of chapter VII, the leading method is not suitable for teaching a robot to do eye movements. Chapter VIII establishes the basic computing power of MCLSs. Chapter IX shows that an MCLS can handle the generalization problem of "negation".

IV.1 AN MCLS PROVIDES SEQUENCES AND GOALS

Section 1.1 describes how an MCLS can provide an internal representation with both goals and sequences in it. The way an MCLS may implement VC is summarised in section 1.2.

IV.1.1 Sequences and Goals

There are three ways for an MCLS to provide an internal representation with sequences in it:

(a) the context of a rule's quintuple may have a sequence of recent stimuli and/or actions in it, for example the sequence of stimulus pairs "1 kg 3 cm, 1 kg 4 cm, 1 kg 2 cm", instead of only the most recent stimulus pair "1 kg 2 cm". An example is given below;

(b) actions in the context of a production can represent earlier sequences of actions and stimuli. For example, an MCLS could enable a robot to say "ONE" to itself, having achieved the sequence of three stimulus pairs in (a). A context could have a sequence of speech

actions in it, for example "ONE, THREE, TEN", each speech action representing a sequence of other stimuli or actions. These actions are sometimes called "auxiliary" actions (Andreae & Cleary, 1976; Andreae, 1977a; 1979b). An example is given below;

(c) the MCLS can organise some of its quintuples into a working memory, enabling it to remember and perform an arbitarily long sequence of actions. For example, an MCLS can simulate a tape memory, storing a sequence of symbols on the "tape" (MacDonald & Andreae, 1981; MacDonald, 1980; Andreae 1980a), as will be explained in chapter VIII.

Other sorts of production that an MCLS can have are discussed elsewhere (Andreae, 1977a; 1981; 1982a), but they are not important here.

I shall now describe several simple examples of MCLSs which a teacher could teach to perform a sequence of movements. The internal representation of the type suggested in (a) above will first be employed. The method suggested in (b) will then be employed in an example. Finally, I will describe a two-rule MCLS that can be taught both sequences and goals. All the MCLSs in these examples will have productions with only one transition in them. The productions have the form <Context, C', Next context, Goal, Val>. My examples will be much more readily explained by having only one transition per production.

Suppose a robot must perform the sequence of actions

+1 -1 +1 -1 +3 -2 +2 then -3

These might be actions for vertical strokes of a spray-painting robot's arm. They might paint an object as it passes along on a conveyor belt, as shown in Figure IV-2, and then bring the arm down to start the next object. There is a microswitch which is switched by any object on the conveyor belt in front of the robot arm. I will assume that the spray

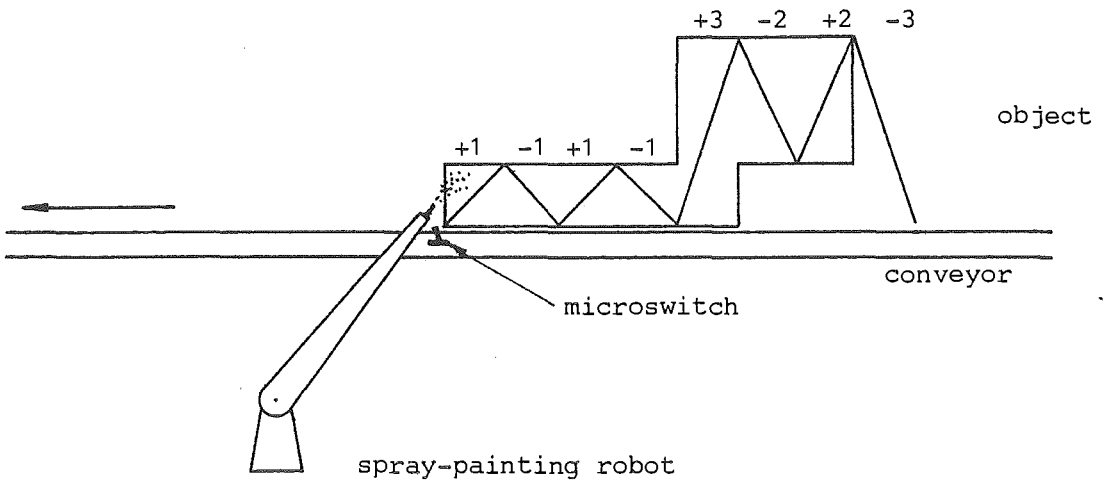


Figure IV-2 A robot performs the sequence of actions +1 -1 +1 -1 +3 -2 +2 -3 in order to paint an object that is moving along a conveyor belt. The arm shown here has only one degree of freedom. An action causes a change in the arm angle over a fixed time interval, at constant speed. For example, the action +2 causes the arm angle to change by 2 degrees over the time interval, while the action +1 causes the arm angle to change by only 1 degree over the same interval. If the robot arm and the conveyor belt move at constant speed then the painting strokes will trace out a triangular pattern on the object, as shown in the diagram. These painting strokes may be an impractical way to paint an object, but that is not important here. The important thing is that an MCLS can be taught to perform sequences of actions like this simple example sequence of spray-painting strokes, as explained in the text.

gun is automatically turned on by the microwswitch.

I will start by describing an MCLS that (i) is equivalent to a GS system, and (ii) cannot be taught the painting sequence shown in Figure IV-2. Consider a one rule MCLS, called PAINT/2S, whose contexts contain both the current arm angle and the state of the microswitch. "2S" in PAINT/2S stands for "two stimuli". Note that such an MCLS is also a GS system, since its contexts are just current conditions, or Cond, of the robot.

Suppose that the sequence shown in Figure IV-2 begins at zero degrees. If the teacher leads the sequence of actions given above, while there is an object in front of the robot arm, then these context-action pairs will be stored,

0 deg Obj ---> +1

1 deg Obj ---> -1

0 deg Obj ---> +3

3 deg Obj ---> -2

1 deg Obj ---> +2

3 deg Obj ---> -3,

where "deg" means degrees, and "Obj" means "object sensed by microswitch". The teacher would then reward the robot. Figure IV-3(a) shows the internal representation formed by this teaching. Now say the teacher puts the arm at 0 degrees, and puts the robot into execution mode. When an object is sensed, the robot would select +3, rather than +1, because in its internal representation +3 is closer to the goal.

Consider an MCLS, PAINT/5S, whose contexts have in them the four most recent arm angle stimuli, and the state of the microswitch, that is five stimuli. The internal representation shown in Figure IV-3(b) would be formed if the teacher led the robot through the painting

Figure IV-3 (a) Internal representation formed in PAINT/2S when the teacher leads the robot through the sequence shown in Figure IV-2. The arrowed lines show the "Next context" of each quintuple.

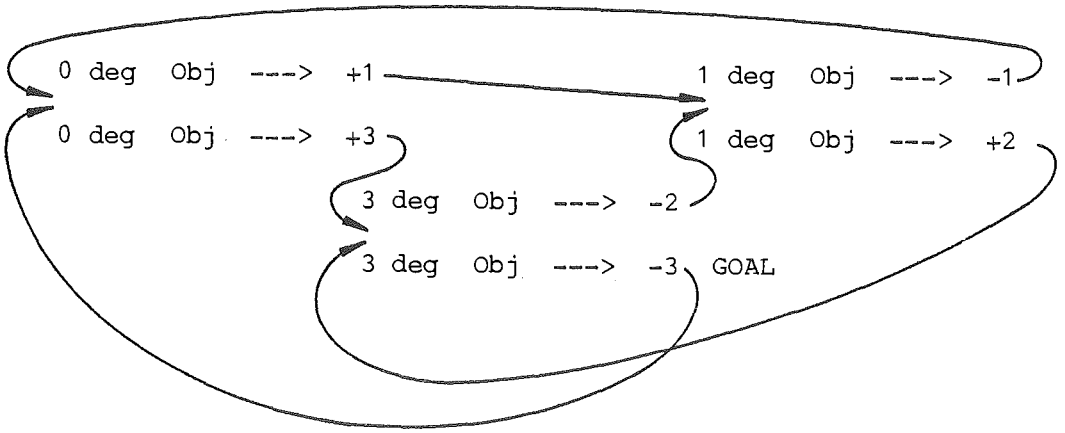


Figure IV-3 (b) The internal representation formed in PAINT/5S when the teacher leads the robot twice through the sequence shown in Figure IV-2. The arrowed lines show the "Next contexts" of the quintuples.

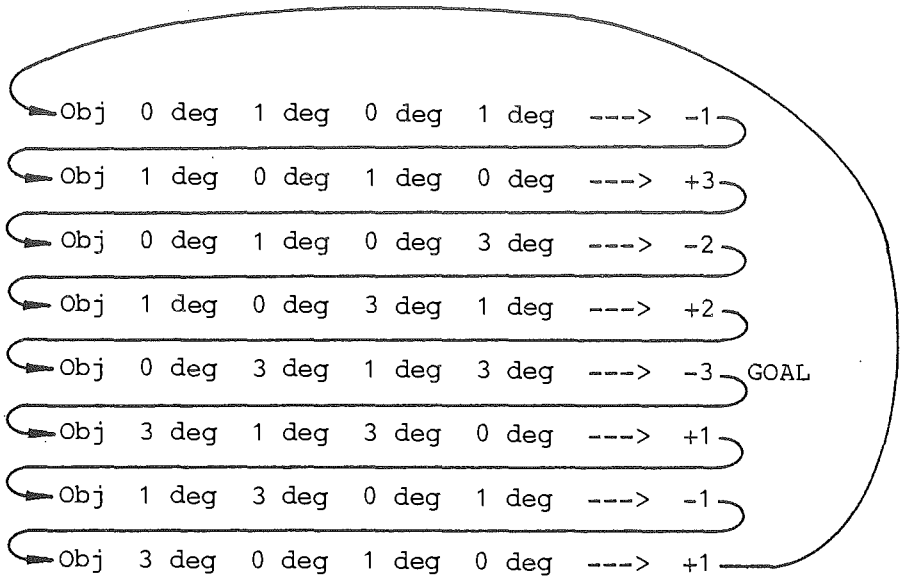
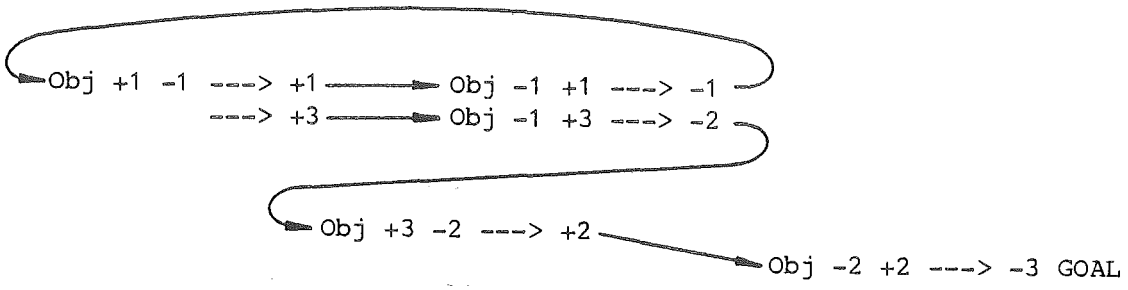


Figure IV-3 (c) Productions formed when the teacher leads PAINT/1S2A through the sequence of movements 1 deg -1 deg 1 deg -1 deg 3 deg -2 deg 2 deg -3 deg, for painting the object shown in Figure IV-2, and rewards it at the end.

```
Obj +1 -1 ---> +1
Obj -1 +1 ---> -1
Obj +1 -1 ---> +3
Obj -1 +3 ---> -2
Obj +3 -2 ---> +2
Obj -2 +2 ---> -3   (this production is rewarded)
```

Figure IV-3 (d) Internal representation created when the teacher leads PAINT/1S2A through the sequence of productions given in (a). Arrowed lines show transitions from one production to another.



sequence twice. If the teacher put the robot into execution mode at 0 degrees, after leading the sequence for the second time, the robot would select +1, then -1, and so on, performing the required sequence of actions. There is only one action "predicted" by each context in Figure IV-3(b).

PAINT/5S shows how contexts with recent stimuli in them can enable an MCLS to be taught a sequence of actions. A context with recent actions in it can also enable sequences to be taught. However, there will not always be enough recent actions or stimuli for the required sequence to be taught.

For example, consider the MCLS, PAINT/1S2A, whose contexts have in them the two latest actions and the state of the microswitch.

PAINT/1S2A cannot be taught the painting sequence. If the teacher led PAINT/1S2A through the sequence of painting strokes, the productions shown in Figure IV-3(c) would be formed. So the internal representation would be the one shown in Figure IV-3(d).

Now, suppose that the teacher leads PAINT/1S2A through the first +1 and the first -1, and then puts it into execution mode. It would perform +3, rather than +1, because in its internal representation, the shortest path to the goal from +1 -1 is through +3. So PAINT/1S2A would not perform the sequence required.

I will now give an example of an MCLS that employs the method (b), given at the start of this section, for internally representing a sequence. Auxiliary speech actions will be taught to the robot. I will suppose that the robot can make speech sounds, and hear speech sounds. The teacher might teach it to say a speech sound, by saying a speech sound himself. [This method of learning speech sounds is the "back-reflex" method (MacDonald, 1979), also called "mimic-speech" (Andreae, 1977a). It is described in chapter VII.]

PAINT/1S2A with speech will be called PAINT/1S2A(Sp). Suppose the teacher leads PAINT/1S2A(Sp) through actions and says speech sounds, in this sequence,

ONE +1 TWO -1 THREE +1 FOUR -1 FIVE +3 -2 +2 -3,

and then rewards the robot. He then does the first two actions again, ONE +1. The internal representation would be the one shown in Figure IV-4. PAINT/1S2A(Sp) will perform the required sequence, saying a different speech word before each of the first five actions.

So an MCLS can be taught sequences. The GS system in section 4 in chapter III is also a single rule MCLS. So an MCLS can also be taught goals. However, I have not yet shown how the same MCLS can be taught

Figure IV-4 Internal representation created when PAINT/1S2A(Sp) is taken through the sequence ONE 1 deg TWO -1 deg THREE 1 deg FOUR -1 deg FIVE 3 deg -2 deg 2 deg -3 deg, rewarded at the end and then taken through ONE 1 deg again.

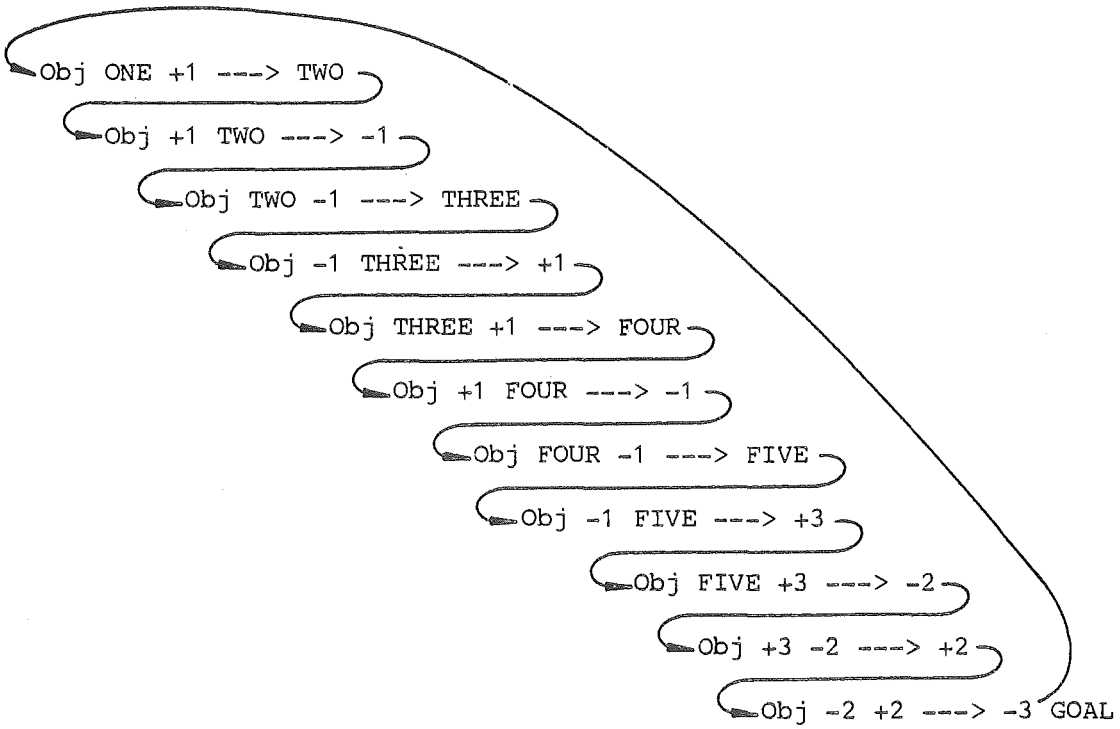
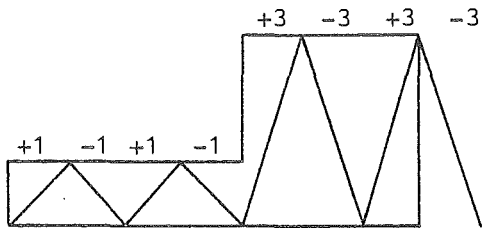


Figure IV-5 A second object for a robot to paint, as well as the one in Figure IV-2. A sequence of painting strokes for painting the object is shown in the Figure.



both goals and sequences. I will show this now, by extending the painting task of Figure IV-2 so that it has in it a goal similar to the goals of the die-casting transfer task in section 4 of chapter III. An MCLS that can be taught the extended painting task will be described.

Suppose that the robot must paint two differently shaped objects, for example the object shown in Figure IV-2, and the one shown in Figure IV-5. Suppose these objects come along on the conveyor alternately. Suppose the robot has no sensory information to say which object is in front of it. The robot must be taught to remember which painting sequence it last performed, so that it knows which sequence to perform next. The teacher might teach this sequence to PAINT/1S2A(Sp),

ONE +1 TWO -1 THREE +1 FOUR -1 FIVE +3 SIX -2 SEVEN +2 EIGHT -3,

then wait for the second object to come along and teach,

NINE +1 TEN -1 ELEVEN +1 TWELVE -1 THIRTEEN +3 FOURTEEN -3 FIFTEEN +3 SIXTEEN -3,

and then reward the robot. Then the teacher could say ONE, lead +1, put the robot into execution mode, and it would perform the required sequence. The internal representation formed in PAINT/1S2A(Sp) would be like the one in Figure IV-4, but longer.

Now I will introduce a goal into the task. Suppose that there are sometimes obstacles in the way of the robot's arm. Suppose that the robot arm has a proximity sensor which tells it whether there is an obstacle above or below its arm. For this illustrative example imagine that there may be obstacles only while the robot's arm is on the way down to zero degrees, when there is no object in front of it on the conveyor belt. Consider the MCLS, PAINT/3S2A(Sp), with contexts which have in them three current stimuli---the current arm angle, the state

of the microswitch, and the state of the proximity sensor---and the two must recently performed actions. I will represent the state of the proximity sensor by "Ab" for obstacle above, "Be" for obstacle below, and "NO" for no obstacle. For this simple example I will assume that there is never an obstacle above and below at the same time.

Suppose the teacher teaches the sequence given above, during which no obstacles occur, and then starts the robot executing the sequence. The internal representation formed in PAINT/3S2A(Sp) during teaching is shown in Figure IV-6(a). Suppose an obstacle is below the arm at 1 degree during execution of the first -3 action, the one after EIGHT. The arm would stop, since no quintuples have been taught for the presence of obstacles. None of the quintuples in Figure IV-6(a) have Be in them. The teacher could then put the robot into leading mode, and lead it through appropriate movements. For example, the teacher could lead the arm through a +1 action and then a -1 action. The first two productions shown in Figure IV-6(b) would be formed, if the obstacle had occurred after the robot painted the object shown in Figure IV-2. The one degree of freedom arm can only wait for the obstacle to move, or try to push it away. Suppose that the obstacle goes away after the teacher leads +1 -1. The teacher must lead -1, to put the robot at 0 degrees and reward the robot. When the next object comes along the teacher must say "Nine", and then lead +1. The context is then "Obj 1 deg NO NINE +1", a production in Figure IV-6(a). The robot would say "Ten" and go on to paint the object.

Now, there is a problem with this teaching of movements in the presence of an obstacle. When the robot performs actions to move away from the obstacle, there is no speech in the current context, as shown in Figure IV-6(b). The robot has "forgotten" which painting sequence

Figure IV-6 (a) Internal representation formed in PAINT/3S2A(Sp). After the teaching the teacher starts the robot executing the task by saying ONE, leading +1, and putting the robot into execution mode.

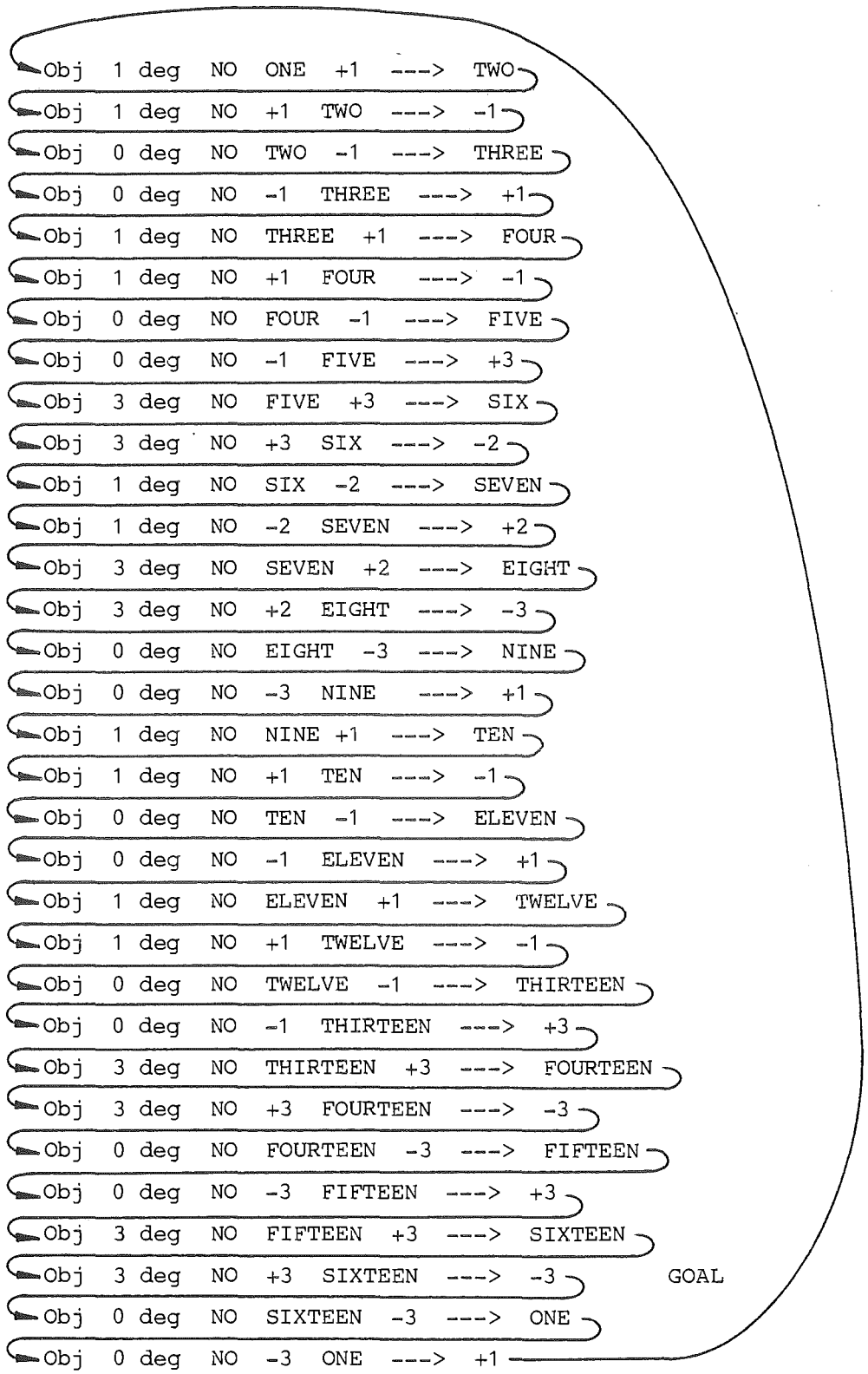


Figure IV-6 (continued) (b) Productions formed when the teacher leads PAINT/3S2A(Sp) through a +1 action, a -1 action, another -1 action, rewards the robot, says "Nine" and leads +1, after the robots stops for an obstacle below its arm at 1 degree. The robot has just painted the first object. "No Obj" means "no object sensed by the microswitch on the conveyor belt".

```
No Obj  1 deg  Be   EIGHT  -3   --->  +1
No Obj  2 deg  NO   -3 +1   --->  -1
No Obj  1 deg  NO   +1  -1  --->  -1    GOAL
Obj  0 deg  NO   -1  -1  --->  NINE
Obj  0 deg  NO   -1  NINE  --->  +1
```

(c) Productions formed when the teacher leads PAINT/3S1A1A(Sp) through a +1 action, a -1 action, another -1 action, rewards the robot, says "Nine" and leads +1, after the robot stops for an obstacle below its arm at 1 degree. The robot has just performed the first -3.

```
No Obj  1 deg  Be   EIGHT  -3   --->  +1
No Obj  2 deg  NO   EIGHT  +1   --->  -1
No Obj  1 deg  NO   EIGHT  -1   --->  -1    GOAL
Obj  0 deg  NO   EIGHT  -1   --->  NINE
Obj  0 deg  NO   NINE  -1   --->  +1
```

it just performed. Suppose the teacher taught +1 -1 -1 ONE +1, when the robot sensed an obstacle at 1 degree, after painting the second obstacle. The productions formed would be like those in Figure IV-6(b), but with SIXTEEN and ONE replacing EIGHT and NINE. There is no speech in the context of the fourth production of Figure IV-6(b) to make sure NINE follows EIGHT and ONE follows SIXTEEN. So if the robot tried to avoid an obstacle itself, after the teacher had taught the productions shown in Figure IV-6(b), it would lose track of the sequence.

Suppose the robot's MCLS is PAINT/3S1A1A(Sp), with contexts of the form arm angle, microswitch state, proximity sensor state, last arm action, last speech action.

The robot would remember the last speech action until another speech action was performed. The most recent speech action stays in the context until another speech action is performed. Any number of arm actions could be performed by the robot, to move away from the obstacle, and the last speech action would remain in the context. The productions formed in PAINT/3S1A1A(Sp) during the teaching of the sequence above are the same as the productions formed by PAINT/3S2A(Sp) shown in Figure IV-6(a). However the productions shown in Figure IV-6(c) would be formed if the teacher led PAINT/3S1A1A(Sp) up +1, down -1, down -1, rewarded the robot, said "Nine", and led +1, when it encountered an obstacle after painting the first object. The speech is held in the context. The robot would remember when to say "Nine" and when to say "one" after encountering an obstacle.

We have now reached the crux of this example: the goal of performing -1 at 1 degree is not being represented properly by PAINT/3S1A1A(Sp). Rather, what is represented by the goal productions

No Obj 1 deg NO EIGHT -1 ---> -1, and

No Obj 1 deg NO SIXTEEN -1 ---> -1,

is that -1 must be performed, a certain speech action having been performed.

This means that (a) the robot's experience of avoiding an obstacle after saying EIGHT cannot be used for avoiding an obstacle after saying SIXTEEN, even if the obstacle is in the same place and behaves in the same way, and (b) the robot's past experience of performing actions in the presence of obstacles cannot be used unless the robot had said EIGHT or SIXTEEN during that past experience. The productions in Figure IV-6(c) have EIGHT in them. However, an MCLS with more than one rule can represent the goal of performing -1 at 1 degree, independently of the speech just used.

Suppose the robot has the two-rule MCLS, PAINT/2R:

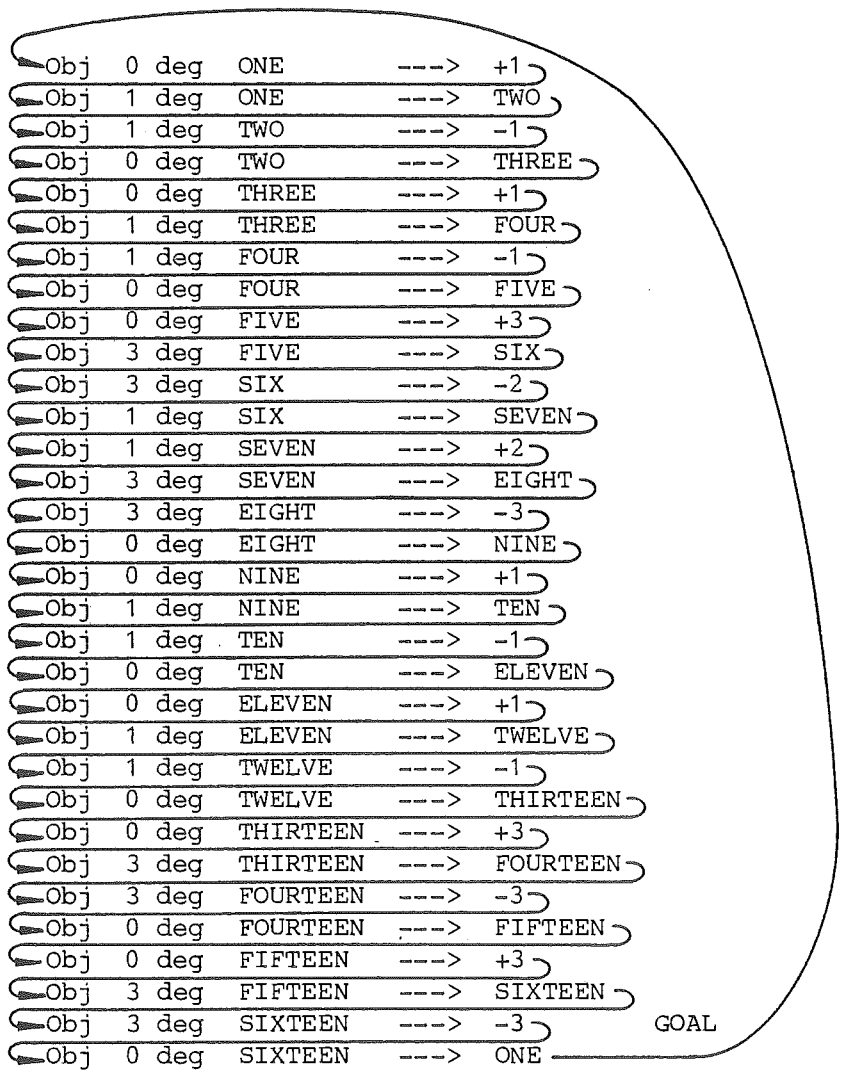
rule 1	form of context:	current microswitch state, most recent speech action, current arm angle
	form of C':	action
rule 2	form of context:	current arm angle, current proximity sensor state
	form of C':	arm action

Suppose the decision procedure selects the current C' from rule 1, if no C' is proposed by both rules. Rule 1 is similiar to PAINT/3S1A1A(Sp). Rule 1 does not have the arm action and proximity sensor state in its contexts, which PAINT/3S1A1A(Sp)'s contexts do.

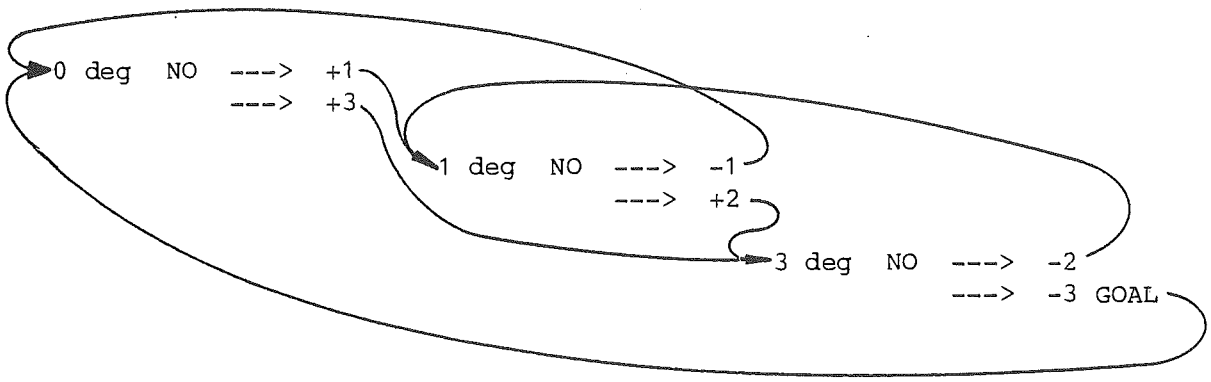
Suppose the teacher teaches the sequence given above for painting the two objects in turn. The internal representation given in Figure IV-7(a) would be formed in rule 1. The internal representation shown in Figure IV-7(b) would be formed in rule 2. [A more detailed explanation of how PAINT/2R learns the first few actions is given in

Figure IV-7 Internal representation formed in PAINT/2R when it is taught to paint the two objects.

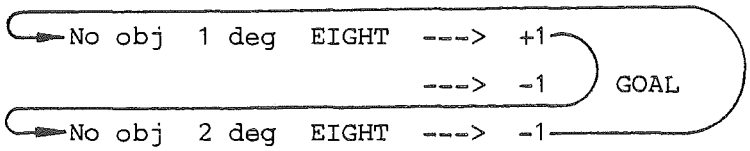
(a) Rule 1



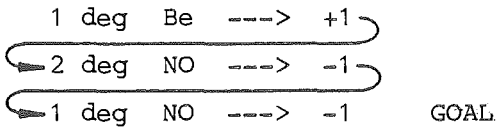
(b) Rule 2



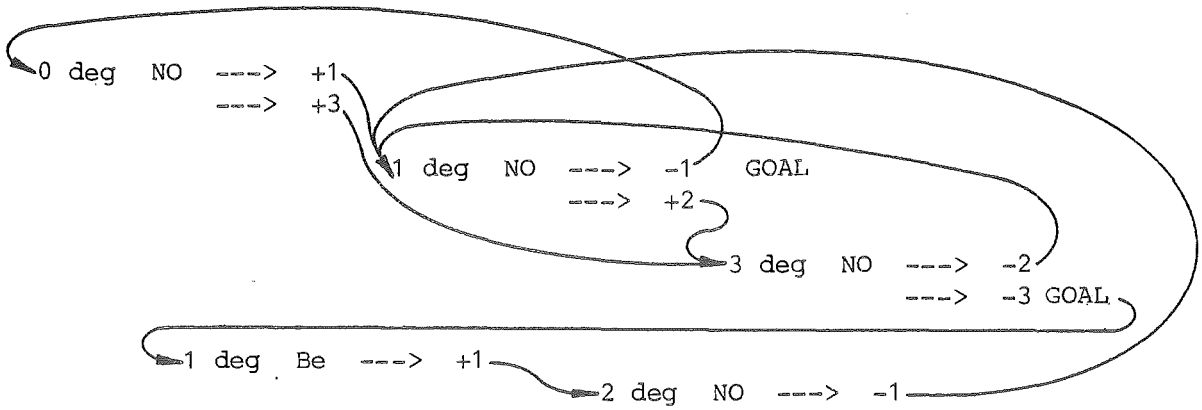
(c) Productions formed in rule 1 when +1 -1 -1 is led after an obstacle is sensed at 1 degree and the robot is rewarded.



(d) Productions formed in rule 2 when +1 -1 -1 is led after an obstacle is sensed at 1 degree and the robot is rewarded.



(e) Internal representation in rule 2 after the productions in (d) have been added to the internal representation in (b).



section 2.2. The explanation is not needed for the present example.] So the robot would perform the required sequence, when no obstacles were present. Rule 1 would predict the action required at each step.

Now suppose that an obstacle is present at an angle of 1 degree, below the arm, just after the robot has painted the first object. There are no productions that apply to this situation, so the robot would stop. Suppose that the teacher leads the actions +1, -1, and the obstacle is no longer sensed, then leads -1 to zero degrees and rewards the robot. The productions shown in Figure IV-7(c) would be formed in rule 1. The productions shown in Figure IV-7(d) would be formed in rule 2.

Figure IV-7(e) shows the internal representation in rule 2. If during the next painting sequence there is an obstacle at 1 degree below the robot's arm, the robot will perform +1 -1 -1 regardless of which object it has just painted. Rule 2 will predict +1 from 1 deg Be. Rule 1 will either predict nothing, from No Obj 1 deg SIXTEEN, or +1 and -1 from No Obj 1 deg EIGHT. So +1 will be performed, as specified by the decision procedure. After +1 -1 -1, the rule 1 context will be Obj 0 deg EIGHT or Obj 0 deg SIXTEEN. So the robot will go on to correctly paint the next object.

Note that the transition out of "3 deg NO ---> -3", to "0 deg NO" in Figure IV-7(b), is deleted, and a new one formed in Figure IV-7(e). The new one is to "1 deg Be". As stated previously, there is only one transition per production in the example MCLSs of this section.

A more difficult MCLS to develop than the sort of example MCLS just described, is one that enables a teacher to teach a robot a task that requires a certain sequence of things to happen, but actually requires different sequences of movements in different situations. For example,

how could a teacher easily teach a led robot a spray-painting task, in a way that (a) prevents the robot from being able to skip parts of a sequence, and (b) enables the robot to select different movement sequences when there are different obstacles encountered during painting? Even in the presence of obstacles the paint must be applied properly. The paint must be applied in a sequence of strokes.

It is known that an MCLS can implement any algorithm that a computer can implement (see chapter VIII; Andreae, 1981; 1980a; MacDonald, 1980; MacDonald & Andreae, 1981). An MCLS has yet to be designed that provides a natural way of teaching complicated algorithms to a led robot. The work in this thesis makes an important step towards this goal by establishing (a) in chapters III, IV and V, that the leading method can be used to teach a robot with an MCLS, and (b) in chapter VIII, that an MCLS can be taught complicated algorithms.

IV.1.2 Implementing VC

In this section I give a very brief summary of how VC could be implemented with an MCLS.

An MCLS could implement VC by having a rule for verbal corrections. The rule has in it productions of the form

IF speech heard is ... THEN DO correction action ...,

for example "Up" ---> UP +1. An example MCLS with VC, PAINT/VC, is described in the appendix.

No transitions or goals need be stored in the verbal correcting productions. No goal-seeking is needed in them. The verbal correcting productions need only select an action when a speech sound is heard. The productions in PAINT/VC are two-tuples, <Context, C'>. The action selected is added onto the action being performed by the robot, as

explained in section 2 of chapter III.

No special correction mode is required. A verbal correction production may be stored whenever the teacher says a speech sound and the robot is either led through an action, or performs an action.

There would be severe problems with VC in section 1.1's MCLSs with speech. This is because VC would interfere with the robot's speech. The teacher's verbal correcting speech would go into the productions of the MCLSs. This would not seem unreasonable to a teacher who is verbally correcting a robot. He would be used to the effect his speech can have on someone who is talking themselves through a task. For example, it is difficult to count objects and have a conversation at the same time. An MCLS should be able to overcome this problem by using actions other than speech to represent sequences. Any actions can be used for this, as long as they are not the type of actions used for doing the task itself. For example, while doing a one-handed task, a robot could count movements on the fingers of its other hand.

II.2 MULTIPLE CONTEXT LEARNING SYSTEMS (MCLSs)

As stated at the beginning of this chapter, a multiple context learning system (MCLS) may comprise one or more context systems, or "rules", a decision procedure, and a sequence of goals. Some examples of simple MCLSs were given in section 1. This section gives a more formal description of the MCLSs that are used in this thesis. No complete formal description can be given of MCLSs because they are an underdetermined class of systems. This is deliberate. Constraints are put on the class of MCLSs only when necessary (see Andreae, 1977a; 1974; Andreae & MacDonald, 1981).[1] Therefore, I will not be able to delimit the class of MCLSs. In section 2.1 I shall give a formal description which includes all the MCLSs employed in this thesis.[2] The terminology of MCLSs is given. Examples of how this terminology applies to PAINT/2R of section 1.1 are given. Section 2.2 gives an example of how this terminology would be used to describe the operation of PAINT/2R. Section 2.3 explains (a) how an MCLS can satisfy the condition of distinguishability, required by section 4.1 of chapter III, and (b) how actions acquired in one situation can be performed in another situation by an MCLS. Section 2.4 briefly states the types of MCLS used in each chapter of the thesis.

II.2.1 Formal Description and Terminology of MCLSs used in this thesis

The terminology for the MCLSs of this thesis is all given in the glossary at the end of the thesis. It will be repeated here, along with more explanation than in the glossary and with examples drawn from the MCLS PAINT/2R of section 1. From now on in the section, all the words that are given as terminology are underlined every time they are

used.

Firstly, four major terms, MCLS, rule, decision procedure and template are given. The rest of the terms are given in alphabetical order.

MCLS

Multiple context learning system. One or more rules, a decision procedure, plus a goal. For example, PAINT/2R is an MCLS. None of the MCLSs in this thesis have more than one goal.

rule

(also called a context system or a network of productions). A rule comprises (a) a network of 2, 3 or 5-tuples, called productions, with one of the five types (i) to (v) below, (b) a template for making the tuples, and, if the productions are of type (iii) or (iv), an algorithm for calculating Val in each production.

(i) <Context, action>

(ii) <Context, action, Goal>

(iii) <Context, action, Next context, Goal, Val>

(iv) <Context, action, <Next context, prob(Next context)>, Goal, Val>

and

(v) <Context, prediction, Goal>

Here I have used "action" instead of the C' used in chapter III. Motor commands, C's, are also called actions. "prob" stands for "probability". Rules of type (i) are just lists of two-tuple productions. Rules of type (ii) are lists of productions, some productions being goals. There is no networking of type (i) and (ii) productions; no paths to goals are stored in the internal representation. There are no transitions making paths from production

to production. [As shown by Andreae (1977a), some forms of this type of rule are still able to perform goal-seeking. A process of "hypothesis formation" is employed. Hypothesis formation is not important for this thesis.]

Rules of type (iii) have "deterministic" transitions. The rule stores one Next context per context-action pair. Therefore, when selecting actions, the rule effectively assumes that an action performed in a context will always cause one particular Next context to occur. Rules of type (iv) are similar to a GS system. They have contexts instead of Conds. Rules of type (v) are similar to those of type (ii). However, type (v) rules can select a prediction---an action, a stimulus, or an action-stimulus---not just an action. It is true that type (ii) rules are also type (v) rules. However, it will be useful to distinguish between rules of type (v) that are also of type (ii), and type (v) rules in general. For example, some of the rules used in chapter VII are type (ii), and some type (v) but not type (ii). Thus I may say that the MCLS of chapter VII has rules of type (v).

For example, rule 1 in PAINT/2R has the form (iii). A network of productions for rule 1 is shown in Figure IV-7(a).

At any instant the internal representation in a type (iv) rule is equivalent to that of a Markov decision process (MDP). Briefly an MDP will be in one of a number of "states", say state i (see Howard, 1960). In each state a number of "decisions" are available to choose from. When a decision, say k, is chosen in state i, the state will change, say to state j. j may be the same as i. The probability of the state changing from i to j, given decision k, is a function of only i, j and k. When a certain decision is chosen in a certain state, "reward" may be obtained. An implementation of an MDP will choose decisions for

obtaining rewards. That is, some "value" will be explicitly or implicitly assigned to each decision of each state.

Thus the internal representation in an MDP may be described by a list of quintuples, <state, decision, <next state, prob (next state)>, reward, value> which has the same form as type (iv) productions.

Note that the states, in the MDP equivalent of a rule, are not equivalent to the states the robot may be in, nor to the states the MCLS may be in. The MDP states are equivalent only to that rule's contexts.

Figure IV-8 shows how the productions of an MCLS are often actually stored. It is important that productions with a common context are stored together. All the predictions of that context are then readily available.

Decision Procedure

A decision procedure selects actions from the combined predictions of the rules on the basis of a weighted majority vote[3]. The weight of a particular prediction depends on (a) the weight or priority of the rule that made the prediction, and (b) the weight or priority of the prediction in the rule[4]. For example, the decision procedure of PAINT/2R is "select the prediction of rule 1 if rules 1 and 2 disagree".

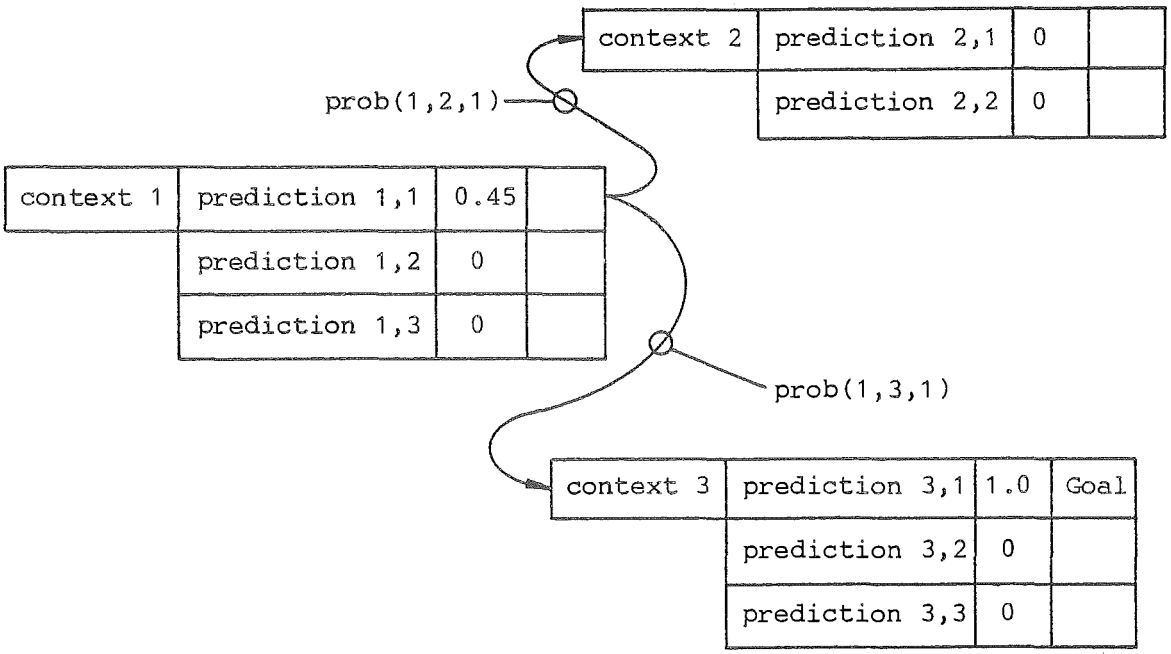
Template

A prescription for forming the context-prediction pairs in productions, from the actions and stimuli of the robot. For example the template for rule 1 in PAINT/2R is "current microswitch state, most recent speech action, current arm angle" for the context, and "action" for the action. A production template is a template for forming productions, a

Figure IV-8 All the productions with the same context are stored together. Once a context is found in LTM that matches the STM context of the rule, all the predictions of that context are immediately available. Transitions---the probability of particular contexts following the occurrence of a particular action and particular context---are stored as shown by the circled connections in the Figure. Prob(1,2,1) is the probability of the next current context being context 2, if, while the current context is context 1, the 1st predicted event of context 1 occurs.

The organization shown in the Figure forms a Markov decision process, explained in chapter VI. A "leak-back" process calculates the goal or reward value of each production. The goal-value is an estimate of the expected future goals or rewards that will be obtained if the production is reached.

For example, if (i) context 3 is the only context with a goal production in the LTM of the rule, and (ii) prob(1,3,1) is 0.5, then the Val of prediction 1 in context 1 is 0.5 x 1.0 x a discount factor. If the discount factor is 0.9 then the Val is 0.45, as shown in the Figure. Prediction 1 would be preferred to other predictions of context 1, if none of them give a higher goal-value. Leak-back is discussed in detail in chapter VI.



context template is a template for forming contexts and a prediction template is a template for forming predictions. The production template comprises the context template plus the prediction template.

In addition the template of a rule must specify whether a rule is a "choice" or a "recency" rule. In some MCLSs only one prediction is allowed for each context. Suppose that there is a production which has context C1 and a prediction P1 in it, in LTM. Suppose that the prediction P2 is learned in context C1. Normally, an additional production C1 ---> P2 would be stored. However, in some rules, the production C1 ---> P1 would be changed to C1 ---> P2. The template of a rule must say which of these two possibilties must be used in the rule. A rule which adds productions is called a "choice" rule, because there may be a choice of predictions to make for a context. A rule which replaces productions in the way just described is called a "recency" rule, because only the most recent prediction is stored in a production with a context. In this thesis, the only MCLS to have recency rules in it is the MCLS of chapter VIII. As explained there, the use of a recency rule is a vitally important part of the simulation of a universal computing machine. The recency rule enables the simulation of a "tape" memory in the MCLS.

Action

A command sent to the robot's body control systems by the main robot controller. Also called a motor command. For example, "+1" is an action learned by PAINT/2R.

Context

A set comprising some of the robot's recent stimuli and actions. A context is formed using a context template for a rule. A context is an ordered subset of the set (recent stimuli) u (recent actions), where "u" indicates set union. For example, "Obj 0 deg ONE" is a context formed by the example context template given under "template", for rule 1 of PAINT/2R.

Goal

A code put into a production to indicate that it is goal production. For example, there is a goal in PAINT/2R. The goal is shown in Figures IV-7 (a) and (b) as just "GOAL".

LTM

Long term memory. The network of productions stored in an MCLS's rules. LTM is used in preference to the phrase "network of productions" especially when the productions have the form (i), (ii) or (v) under "rule". In these cases the productions are not networked together.

Prediction

Of a production: an action, stimulus, or action-stimulus that is the second member of a production. It is called a prediction because in the sense that the prediction previously occurred after the context was formed, the context "predicts" that the prediction may occur again, should the context itself recur. For example, the prediction of the production "Obj 0 deg ONE ---> +1" is +1.

Of a rule: the prediction of the production in the rule whose context matches the current context of the robot. For example, when

the current rule 1 context is "Obj 0 deg ONE", the prediction of the rule is +1, because +1 is the prediction of the production "Obj 0 deg ONE ---> +1".

Priority

The weightiness of a rule's predictions in the decision procedure. For example, rule 1 has a higher priority than rule 2 in PAINT/2R, since the prediction of rule 1 will be selected in favour of that of rule 2, if the two rules disagree.

Stimulus

Preprocessed information from the robot's sensors. For example, the state of the microswitch, which is part of the contexts of rule 1 in PAINT/2R.

STM

Short term memory. The current contexts of the robot. For example, the STM of PAINT/2R is "Obj 0 deg ONE, 0 deg NO", when the robot controlled by PAINT/2R (a) senses an object to be painted, (b) has just performed the action ONE, (c) has its arm at zero degrees, and (d) senses no obstacles. Sometimes only a list of actions and stimuli is given; for example "Obj, 0 deg, ONE, NO". This list is unambiguous, since the contexts can be deduced from it.

Transition

The "Next context" part or "Next context, prob (Next context)" parts of a production. For example, the transition out of the rule 1 production "Obj 0 deg ONE ---> +1" in Figure IV-7(a) is to the context "Obj 1 deg ONE".

Val

A measure of how far from a goal a production is. The greater the Val of a production is, the closer that production is to the goal. For example, the production "Obj 3 deg SIXTEEN ---> -3" in Figure IV-7(a) would have the highest Val of the productions shown in the Figure for rule 1 in PAINT/2R. The production immediately above it would have the next highest Val, and so on.

One can refer to contexts, predictions and productions according to the rule they come from. For example, one might call rule 1 of PAINT/2R the "sequence" rule, since it is being used for teaching the robot a sequence of actions. Rule 2 might be called "stimulus" rule, since it has only stimuli in its contexts. The two names distinguish the two rules, enabling one to discuss them. For example, "Obj 0 deg ONE" would be a sequence context, and "0 deg NO" a stimulus context. One may also say that, for example, the sequence rule does this or does that, since "rule" refers to the forming of contexts and productions, and the selection of predictions. For example, one could say that the sequence rule of PAINT/2R remembers which part of a sequence the robot is performing by counting the actions it performs (ONE ... TWO ... THREE ... et cetera).

The main cycle of operations of an MCLS is given below. The step numbers 1 through 4 are shown on Figure IV-1.

1. Update STM with actions performed or learned and with stimuli received.

2. Find the predictions of the rules. That is, for every rule, get the prediction of each production whose context matches that rule's context in STM.

3. Apply the decision procedure to the predictions, to select an action to perform.

4. Perform the action.

5. Goto 1.

The actual implementation of these steps in a real MCLS is more complex than stated here. The implementation is different for different MCLSs.

II.2.2 Example Use of Terminology

Consider the teaching of PAINT/2R, by the teacher leading and speaking to produce the sequence given in section 1, for painting the two objects on the conveyor belt. I will describe what happens during the teaching of the first five actions:

ONE +1 TWO -1 THREE.

The MCLS begins operation with no productions in it. Its LTM is empty of productions. Suppose that when the teacher switches the robot on, (a) there is no object in front of it on the conveyor belt, (b) its arm is at zero degrees, and (c) there is no obstacle sensed near its arm. Rule 1 has the context template "current microswitch state, most recent speech action, current arm angle", while rule 2 has the context template "current arm angle, current proximity sensor state". So once a stimulus has been received by the MCLS from the microswitch, arm and proximity sensors, the contexts will be "No Obj 0 deg" for rule 1 and "0 deg NO" for rule 2. STM is these two contexts.

The teacher then says "One". Now, when the MCLS receives the ONE, no productions are stored. No rule 2 production is stored because the prediction template for rule 2 is "arm actions", while ONE is a speech

action, not an arm action. No production is stored in rule 1 because there is not yet a complete context for rule 1.

The ONE will be put into STM, in the context of rule 1. So STM is now

No Obj 0 deg ONE

0 deg NO.

After a time, an object comes along on the conveyor belt, and trips the microswitch. Immediately this happens, STM becomes

Obj 0 deg ONE

0 deg NO.

The teacher, on seeing the object, leads a 1 degree movement, leading the spraying arm of the robot over the object. The spray gun is turned on by the microswitch when the object comes in front of the robot arm. A +1 action is learned as a result of 1 degree being led. That is, the two productions

Obj 0 deg ONE ---> +1, and

0 deg NO ---> +1

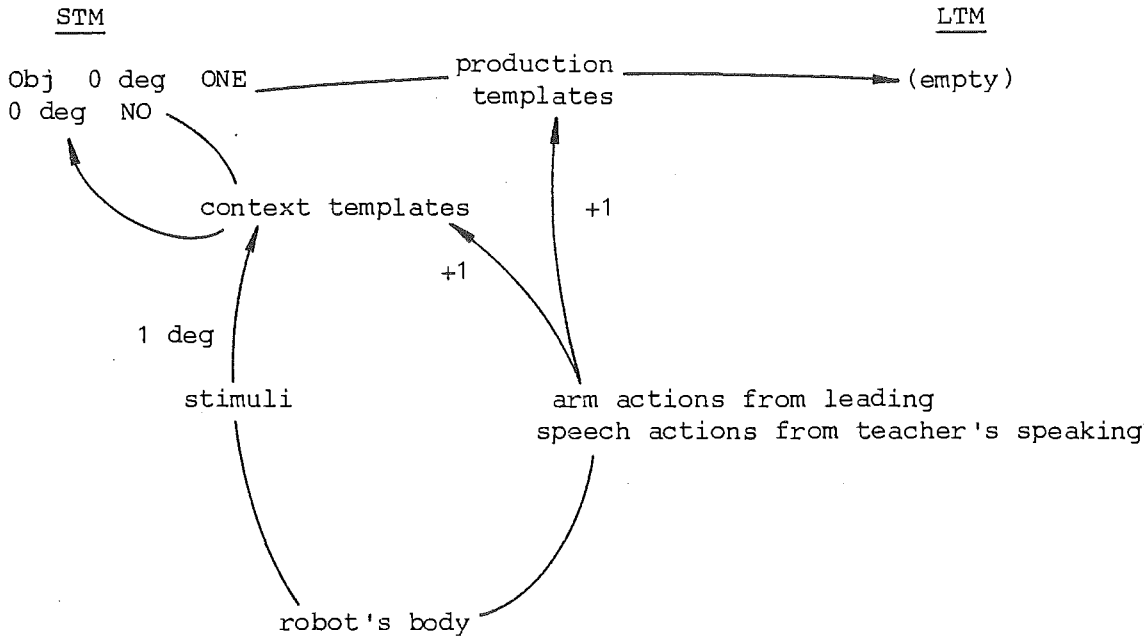
are stored in LTM. STM is updated with the action +1 and with the new arm angle 1 degree. Suppose that there is still no obstacle. The STM will be Obj 1 deg ONE, 1 deg NO.

Figure IV-9(a) shows the STM and LTM before (a) +1 is stored in productions in LTM, and (b) STM is updated with +1 and 1 degree. Figure IV-9(b) shows STM and LTM after both (a) and (b) happen.

The teacher then says "Two", causing a production to be stored in LTM for rule 1, as shown in Figure IV-7(a). TWO is put into STM. Figure IV-9(b) shows the STM and LTM just before TWO is stored in LTM and put into STM. The teacher leads -1 degree, causing two productions to be stored, which predict -1. He then says "Three", and so on.

Figure IV-9 Operation of the MCLS PAINT/2R, of section 1.1

(a) The teacher has said "One", before an object came along the conveyor belt. +1 is led, from 0 degrees to 1 degree. +1 is about to be stored in two productions, in LTM. (b) shows the two productions in LTM. There is no obstacle sensed.



(b) STM and LTM after +1 is led, in (a). The teacher has just said "Two".

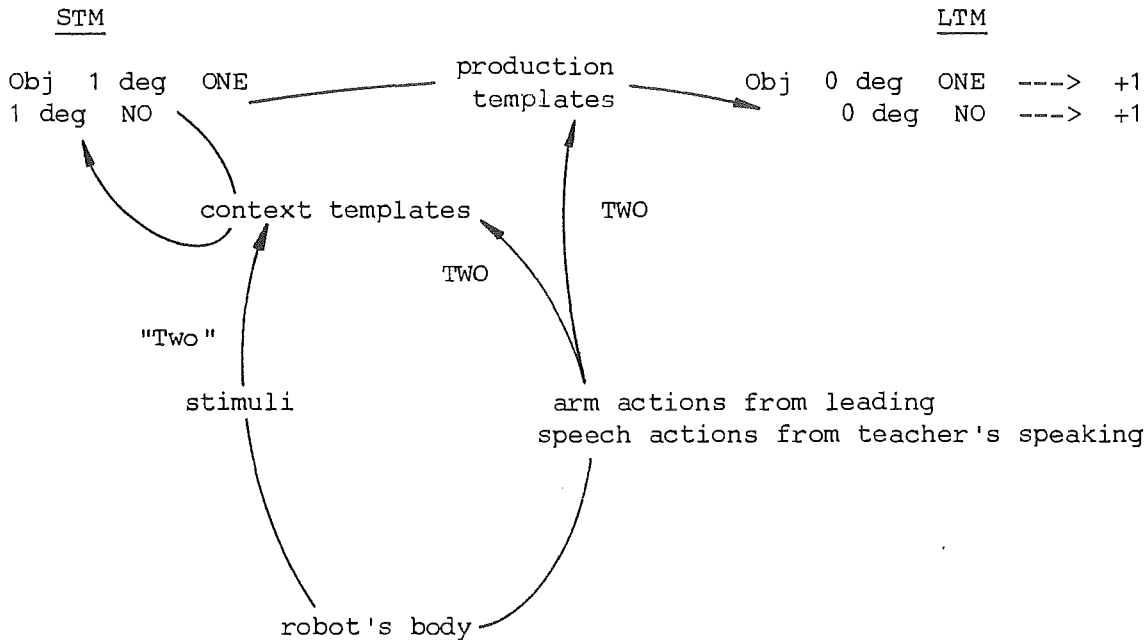


Figure IV-9 (continued)

(c) The selection of +1 by PAINT/2R, once all the productions for the painting sequence have been learned. Both rules predict +1, so +1 is performed. Vals have been omitted from the productions in LTM. The Val of +3, predicted by rule 2, would be greater than the Val of +1 predicted by rule 2. The combined prediction of +1 by both rules outweighs the higher Val that +3 has in rule 2.

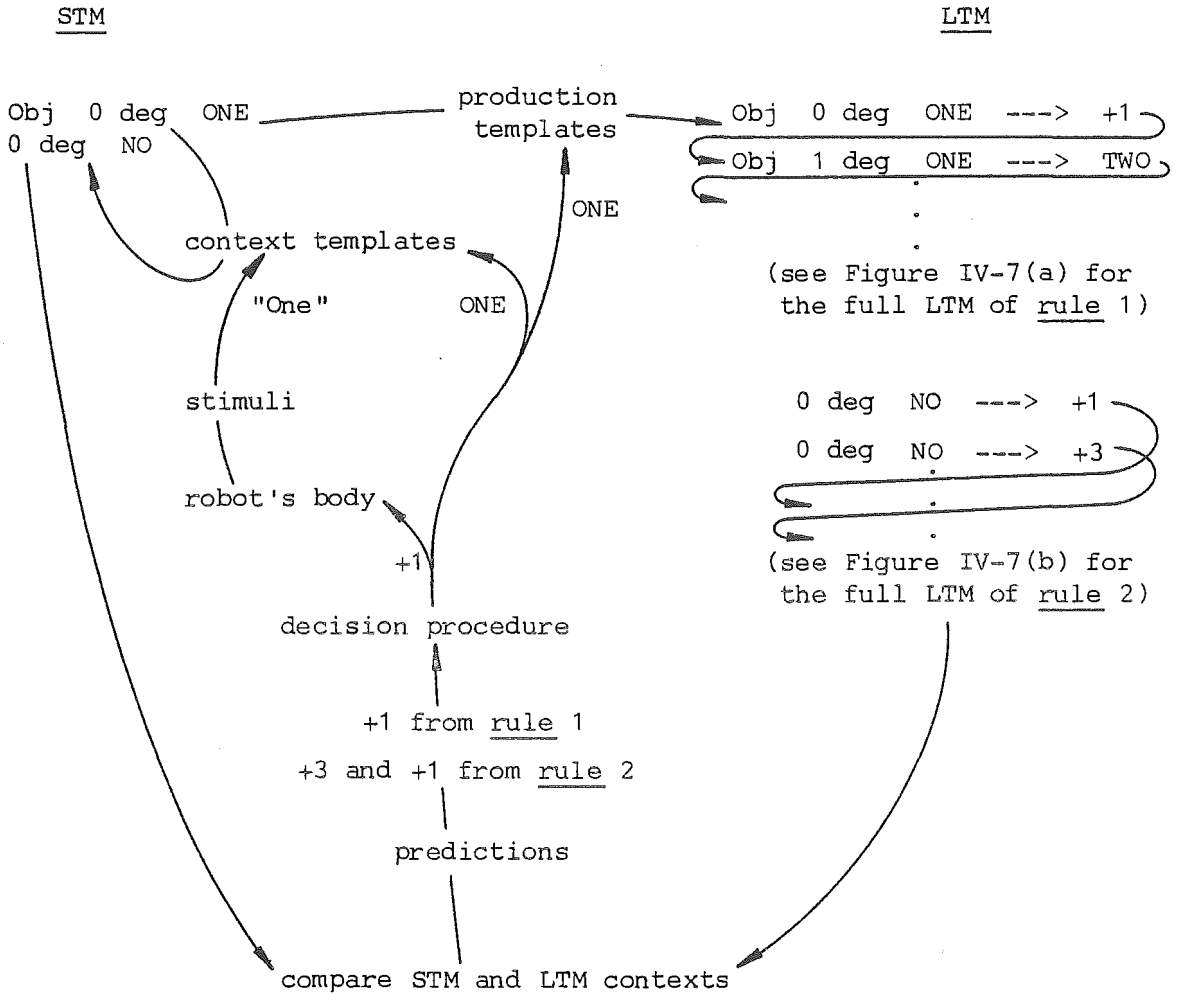


Figure IV-9(c) depicts the selection of +1 once all the productions have been learned, and the robot's STM is

Obj 0 deg ONE

0 deg NO.

These two contexts are compared with the contexts of the productions in LTM. The predictions of productions with matching contexts are sent to the decision procedure. The three productions "Obj 0 deg ONE ---> +1", "0 deg NO ---> +3" and "0 deg NO ---> +1" match contexts in STM. The Val of +3 is higher than that of +1, in rule 2. This is because there are fewer transitions to the goal via +3, than via +1, as shown in Figure IV-7(b). Now rule 1 has a higher priority than rule 2, that is rule 1's predictions have a greater vote in the decision procedure than rule 2's. So +1 will be performed. +1 is sent to the robot's arm control system. If instead rule 1 had a lower priority than rule 2 in the decision procedure, then +3 might be performed. The Val of +3 would have to be much greater than the Val of +1 in either rule, for the +3 to be performed. Only one rule predicts +3.

IV.2.3 Properties of MCLSs

In this section two properties of MCLSs are discussed. These two are important for the way an MCLS implements the leading method.

Firstly, in section 2.3.1 I will discuss the way an MCLS can satisfy the condition of distinguishability required by section 4.1 of chapter III. The contexts of an MCLS must distinguish the different environmental conditions of a task. The contexts must also distinguish the achievement of a goal from the non-achievement of a goal, if the goal is to be properly represented in the MCLS.

Secondly, in section 2.3.2 I will describe how an MCLS can generalize actions from one situation to another. This can enable actions taught by the teacher for one situation to become available in other situations. It is possible for actions to be generalized from one situation to a very different situation. This may occur by successive generalization across a number of similar situations.

IV.2.3.1 Distinguishability I explained above in section 1.1. that an MCLS can have actions in STM which represent earlier actions and stimuli of the MCLS. I also mentioned that an MCLS can implement and use a very large "tape" in its LTM, as a working memory. So in principle there is no real limit to the information that an MCLS can use to distinguish different situations in a task. In practice there may be problems in teaching an MCLS to use this information. What I have just said in this paragraph also applies to the distinguishing of goal achievement from goal non-achievement, since a goal is a particular situation.

The information required for distinguishing situations of a task may include information about the timing of stimuli and actions. It is not yet certain how an MCLS should best have timing information [see section A.4.5 in chapter V, and also Palfi (1976; 1977a; 1977b) and MacDonald (1981)].

IV.2.3.2 Generalization of Actions It is possible to design an MCLS that allows actions to be transferred from one situation to another. Most MCLSs do this (see Andreae, 1977a; 1979c; Andreae & Andreae, 1979; MacDonald, 1982a; MacDonald & Andreae, 1981)[5]. For example, suppose an MCLS has two rules, with these production templates:

rule 1 last speech stimulus ---> arm actions

rule 2 last arm angle ---> arm actions

Suppose that the robot has been led through the action +10 from 30 degrees, after the teacher said "ten". The productions 30 degrees ---> +10 and "ten" ---> +10 would be stored. Suppose the teacher says "ten" when the arm is at 50 degrees. Then rule 1 will predict +10. If +10 is performed then the production 50 degress ---> +10 will be stored. The +10 has been generalized from the situation "ten", 30 degrees to the similar situation "ten", 50 degrees. [Note that in this example the speech is not being used for verbal correction; it is simply one of the stimuli of the MCLS. The +10 in "ten ---> +10" is an arm action, not an arm correction action.]

As pointed out in chapter III, sometimes the action led by a teacher in a situation will not be the action that produces the required movement in that situation. A different action may produce that movement. An MCLS can enable actions to be led in other "non-task" situations and transferred to the task situations by a process of generalization. Once an action has been learned in one situation, only part of that situation has to be repeated for that action to be performed again. That is, a new situation has only to be similar enough to a previous situation for an action learned in the previous situation to be repeated in the new one. Then the action will have been transferred to the new situation, including the new, dissimilar, part of the new situation. In the example in the last paragraph, the action +10 was transferred from the situation "ten" 30 degrees, to the similar situation "ten" 50 degrees, by generalization from "ten", across from 30 degrees to 50 degrees. The action has been transferred from one situation to another by being generalized from the similar part of the situations, across the

dissimilar parts of the situations.

The generalizing power of an MCLS depends on the way the MCLS represents parts of situations in its contexts. The generalizing power will be very weak if a small change in the situation results in such a massive change in the representation that there is not enough similarity for actions to be repeated.

The generalizing ability of MCLSs is important to the leading methods discussed in chapter III for two reasons.

Firstly, suppose leading has taught a robot an incorrect action, say C, for producing a movement, say M. A robot with an MCLS may be able to perform the correct action C', without the teacher verbally correcting C to C'. It could do this if C' had been taught in another situation, and could be transferred to the situation in which M must be produced.

Secondly, a robot with an MCLS may be able to correct its own actions. Correction actions taught by a teacher in one situation might be transferred to other situations, enabling the robot to correct itself in those other situations.

The appendix explains these two possibilities in more detail.

IV.2.4 MCLSs Used in this thesis

In this section the sorts of MCLS used in each chapter are summarised. Then the differences between rules and GS systems are briefly stated.

Chapter IV

The MCLSs discussed in this chapter have rules of type (iii) or of type (i). The only MCLS with a rule of type (i) is PAINT/VC of section 1.2. PAINT/VC is an MCLS for a robot that can be verbally corrected.

Chapter V

The MCLS in chapter V has a rule of type (i) and a rule of type (iv). The templates for the two rules of chapter V are

ANGLE rule: arm angle, arm velocity ---> arm action

ACTION rule: arm action, arm velocity ---> arm action

ANGLE rule is of type (iv). ACTION rule is of type (i). The MCLS in chapter V is an example of an MCLS that employs leading.

Chapter VI

The MCLSs discussed in Chapter VI have rules of type (iv). In the chapter I explain how the Vals in the productions can be calculated.

Chapter VII

The MCLS in chapter VII has rules of type (v). There are several rules. This MCLS, "PURR-PUSS", has been used before for numerous demonstrations of the abilities of MCLSs (Andreae, 1977a; 1979a; Andreae & Cleary, 1976; Andreae & Andreae, 1978; Andreae & Andreae, 1979). In chapter VII I demonstrate that an MCLS can learn to "voluntarily" perform actions that are initially programmed into it as reflexes. PURR-PUSS employs "hypothesis formation" for goal-seeking. The goal-seeking of PURR-PUSS by hypothesis formation is not important for the demonstration in this chapter.

Chapter VIII

The MCLS in chapter VIII has rules of type (i). The MCLS is taught to execute an algorithm. No goal-seeking is employed. The algorithm taught enables the MCLS to simulate a universal computing machine that is called a "Turing machine", which is explained in that chapter. As stated under "template" above, the MCLS in chapter VIII is the only one

in this thesis which has recency rules. All its rules are recency rules.

Chapter IX

The MCLSs in chapter IX have rules of type (i). There are three MCLSs employed in the chapter. They differ only in their decision procedures. The MCLSs are taught to execute three slightly different algorithms, one for each MCLS. Each of the three algorithms is a slightly different way of handling the "negation" problem. The negation problem is the problem of a robot doing something positive when a certain condition is absent. The negation problem is explained in more detail in chapter IX.

The main difference between rules and GS systems is that rules have contexts instead of Conds.

None of the rules in this thesis have a sequence of goals in them. Rules in an MCLS do not need to have a sequence of goals recorded in them as a GS system does in Figures III-6 and III-10. A robot with an MCLS in it can learn and use auxiliary actions to remind itself which part of a task it is performing (Andreae & Cleary, 1976; Andreae, 1977a; 1979b). The example MCLS PAINT/1S2A(Sp) , explained in section 1.1, has actions in its contexts which represent parts of a task. The speech action ONE "reminds" the MCLS that it has performed only the first +1 -1, but not the second +1 -1. It may still be useful to a teacher for there to be several goals he can set. The MCLS depicted in Figure IV-1 can have a number of goals set and activated.[6]

Robots with GS systems would have external controls that the teacher uses to put the robot into either training or execution mode. An MCLS can implement leading without an external control for changing the

robot from execution mode into training mode, or back again, as demonstrated in chapter V (see also MacDonald, 1981; 1982a). The execution-training mode changing may be automatic. If the robot stops performing actions, the teaching mode is entered automatically, after a "time out" period. The robot may select an action while in training mode, causing execution mode to be entered automatically, and the action to be performed.

The formal description of MCLSs in section 2.1 did not say that during leading productions are stored without force information in them. As stated at the beginning of section 2, the description in section 2.1 encompasses the MCLSs of this thesis, but the class of MCLSs is underdetermined. The only MCLS that employs leading in this thesis, the arm-robot MCLS of chapter V, has no force stimuli. So the difference between force stimuli during leading, and force stimuli during execution, does not arise in the formal description of MCLSs in this thesis.

The way non-force productions could be stored, and the way forces could be put in during execution, were described in section 4 of chapter III, for GS systems. These methods might be employed in rules of an MCLS; non-force contexts being stored during leading and forces being put in during execution.

However, it may be sensible to have non-force productions only in robots that collect a limited amount of sensory information. Robots of the future are bound to have good vision and bodily touch sensing systems, and a comprehensive system of proprioception. They will sense in many different ways the difference between being led through movements, and executing movements. It would not be possible for the designer of the robot to have the differentiating types of stimuli left

out of the productions stored during leading, without leaving out nearly everything. Much of the robot's sensory information would change when the robot changed from execution to leading, or back. MCLSs have another way of overcoming the difference between leading and execution. An action may be transferred or generalised from the situation in which it was led, to the situation in which it must be performed. Generalisation was discussed in section 2.3.2. An action can be transferred from one situation to another when the situations are similar. Other methods of action transfer are discussed in the appendix. I said above that a robot designer would find it difficult to eliminate leading-execution differentiating stimulus types without eliminating nearly all types of stimulus. Nevertheless, there may still be many stimuli in common between the leading of a <u>particular</u> action, and the execution of that action. So action transfer would be possible between the similar leading and execution situations.

IV.3 PREVIOUS WORK ON MCLSs AND LEADING

The leading of tasks, and the way an MCLS learns tasks by being led through them, are investigated also in MacDonald (1981; 1982a). In MacDonald (1981) I emphasize the need for a robot to "obtain and respond to information about the environment", so that the robot can overcome the fact that incorrect information is given during leading. In particular, I point out that the teacher, during his leading of a robot, affects the robot-environment interaction. The interaction during leading is different from the interaction during execution. This is another way to say that (a) led actions may need to be corrected, and (b) the transitions stored during leading may not be the actual transitions that occur during execution. Both (a) and (b) were explained in chapter III, (a) at the start of the chapter and in section 2, and (b) in section 4.2.

In MacDonald (1982a) I emphasize the need for led actions to be "transferred" from situations in which they have been led, to situations in which they have not been led. It is not always possible to lead an action in a situation. I also emphasize the need for the dual control that was discussed in section 4.2 of chapter III. Once again, the point is made that the information the robot obtains during leading may not be correct for execution. So the robot must employ some dual control method of selecting actions to perform, so that it may correct its "information".

In MacDonald (1979) the notion of "environment-forcing" is introduced. Environment-forcing is an early suggestion of how leading might be implemented with an MCLS. The robot resists a teacher leading it. Some examples of control systems which might implement it are

given. A simple test is reported. It shows that a robot body control system employing environment-forcing would be stable. Chapter VII explains these suggestions and tests in detail.

IV.4 CONCLUSION

A multiple context learning system (MCLS) comprises one or more rules, a decision procedure and a goal. The terminology for MCLSs is given in section 2.1. A rule comprises a network of productions, plus a template for forming the productions. The decision procedure selects actions from the combined predictions of the rules. An MCLS can enable both goals and sequences to be taught to a led robot. As explained in chapter III, a goal-seeking (GS) system enables only goals to be taught, and a production system of corrections (PSC) enables only sequences to be taught. An MCLS can implement verbal correcting (VC). Some form of MCLS may be suitable for advanced led robots.

NOTES FOR CHAPTER IV

1. Briefly, the design strategy in designing an MCLS for a robot has been to (a) find an important class of problems which the current design cannot handle, and then (b) modify the design so that the additional class of problems is handled, as well as all the classes of problems handled by the previous design (Andreae, 1977a; 1974).

2. Some elaborations an MCLS may have are: novelty (Andreae, 1977a); apathy (Andreae, 1980a); boredom (Andreae, 1980a; 1980b); hypothesis formation (Andreae, 1977a); disapproval (Andreae, 1977a); recursive contexts (Andreae, 1980a; 1980b; 1982a); two-level contexts (Andreae, 1973); decisions based on only partial context matches (Cleary, 1980a; 1980b); a second multiple context (MC)---either MC-There & Then (Andreae, 1980a; 1982a) or MC-Where & When (Andreae, 1982b)---which operates "ahead in time and space" of the main MC, MC-Here & Now. MC-There & Then and MC-Where & When are explained briefly in section 4 of chapter VI.

3. MC-There & Then and MC-Where & When (see note 2) select stimuli, as well as actions, enabling the robot to "look-ahead" of its present time and place.

4. The weight of a rule's predictions could be made to depend on how many predictions it made, or on the number of predictions being made by other rules. Such a priority scheme has never been used in an MCLS.

5. For example, Andreae (1977a) describes how some of what is learned when the task of counting objects is taught, can be transferred to the task of counting "beads".

6. Goals may not be activated in a sequence that has been set by the teacher. For example, goals might be activated by "drive" mechanisms in the robot's body. Deutsch (1960) has discussed multiple goals, or "drives", in humans and animals.

APPENDIX FOR CHAPTER IV

IV.A.1 Example MCLS with VC

Suppose a robot has the two-rule MCLS PAINT/VC, which has these two types of production,

second last arm action, last arm action ---> arm action (Arm type)

speech word ---> correction arm action (VC type)

Suppose the teacher leads a 1 deg movement after saying "UP", and then a -1 deg movement after saying "DOWN". That is the teacher says the words and leads movements in this sequence,

UP 1 deg ... DOWN -1 deg

These two VC productions would be stored,

UP ---> +1 DOWN ---> -1.

Now suppose the teacher leads this sequence of movements,

1 deg -1 deg 2 deg -2 deg

and then rewards the robot. The internal representation in the Arm type productions would be,

+1 -1 ---> +2 ————————> -1 +2 ---> -2 GOAL

Suppose that the teacher actually wants the second two movements to be 3 deg and -3 deg, instead of 2 deg and -2 deg. Say that the actions +3 and -3 are required for producing 3 deg and -3 deg, respectively. When the robot executes the +2 the teacher says "UP". This causes +1 to be added onto +2, and also the production +1 -1 ---> +3 to be stored. The teacher says "DOWN" when the robot executes -2, causing -1 to be added to the -2, and the production -1 +3 ---> +3 to be stored. The teacher then rewards the robot. The internal representation in the

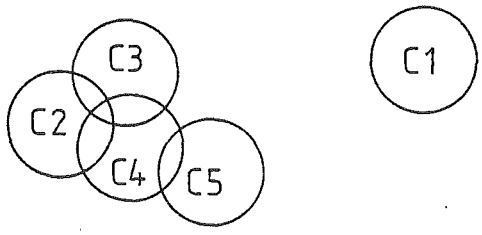
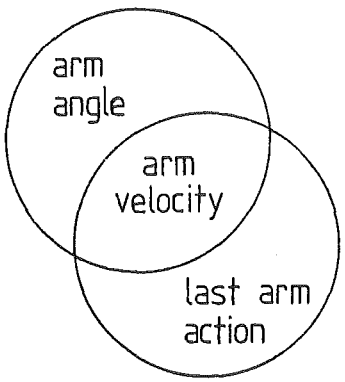


Figure IV-10 Contexts represent parts of situations.

(a) Contexts C2, C3 and C4 overlap in their representation of situations. Context C5 overlaps with context C4, but is independent of C2 and C3. Context C1 is independent of C2, C3, C4 and C5.



(b) These two contexts overlap because they both contain "arm velocity". A situation is characterised by the arm angle, arm velocity, and the last arm action.

Arm type productions is now

+1 -1 ---> +2 ————————> -1 +2 ---> +2 GOAL

+1 -1 ---> +3 ————————> -1 +3 ---> +3 GOAL

Having performed +1 and -1, the robot will have no way to choose between +2 and +3. If the robot chooses +2 the teacher may punish it, by pushing a reward cancelling button. After that the robot would perform +1 -1 +3 -3, as required.

IV.A.2 Generalization

An MCLS represents situations by multiple context (MC), a group of contexts, as explained in section 2. Each context represents a part of a situation. As depicted in Figure IV-10, contexts may overlap in their representations of parts of a situation. For example, suppose

that a MCLS has two rules, whose context templates are:

arm angle, arm velocity (ANGLE rule)

arm action, arm velocity (ACTION rule).

Thus the robot's situations are characterised by the arm angle, the arm velocity, and the last arm action performed or led. For example, "40 deg Stopped" is an ANGLE context with the arm angle "40 deg" and the arm velocity "Stopped" in it. "Stopped" is sometimes shortened to "S". [In fact the example arm-robot MCLS of chapter V has these context templates.]

When an MCLS is led through an action in a situation, productions will be stored which have that action as a prediction. For example, if the ANGLE context is 40 deg S when the action +10 is led, then the production 40 deg S ---> +10 will be stored. The contexts of the productions will be contexts that are in the current MC of the robot when the action was led. Later, if any of those contexts recur, then the productions will be recalled and the action predicted. For example, if later on the context 40 deg S is the current ANGLE context, then +10 will be predicted. If the prediction is stronger than competing predictions for other actions then the action will be performed. In addition, any contexts which did not predict that action, but which have a prediction template that specifies that type of action, will have productions formed with that action. For example, suppose the current ACTION context +20 S when the +10 is performed as a result of the prediction from 40 deg S. Suppose further that there is no production +20 S ---> +10 stored in LTM. Then when the +10 is performed that ACTION production will be stored. The action +10 has been generalized from the recurring context (40 deg S), across the new context (+20 S).

The transferring or generalizing properties of an MCLS have been discussed by Andreae (1977a; 1977c). Section A.2.1 discusses the generalizing of actions from one situation to another, in an MCLS that has no verbal correcting (VC). Section A.2.2 discusses the generalizing of correction actions from one situation to another.

IV.A.2.1 Action generalization: no VC Now, suppose a robot is expected to be able to use its previous experience to provide the actions to perform for a task, rather than the teacher being relied on to provide the required actions. So assume that there is no VC. One might make the strong requirement that the robot be able to perform any action led in any task situation, in any other task situation. I shall discuss this strong requirement, and then relax it in various ways. Rather than suggesting that such a strong requirement be enforced, I will use it as a way of discussing the generalizing power of an MCLS, and the different ways of generalizing actions.

This is a condition sufficient to enable an MCLS to satisfy the requirement that it be able to perform any action led in any task situation, in any other task situation:

> A certain number of rules which have the led actions in their prediction templates must be independent of the task conditions. The number of these task-condition-independent rules must be high enough for them to cause an action to be predicted without agreement from the other, task-condition-dependent rules.

Then the task-independent contexts of the task-independent rules can predict actions which were acquired in one task situation, in any other task situation. For example, suppose an MCLS has a rule that has priority over all other rules. If that rule predicts an action then that action is performed, regardless of the predictions of other rules.

Suppose further that the context template for that rule is "latest speech heard". Then the teacher can say a speech sound when the robot is led through an action, causing a production to be formed. He can then say the speech sound in any task situation, causing that action to be performed by the robot. This is similar to VC, but not the same because the actions predicted by speech are actual actions, not correction actions.

Now speech can be used in the way just suggested for transferring actions from one task situation to another task situation only if the task itself requires no speech sounds to be made. If it does then speech is not independent of the task and so cannot be used for producing actions in any task situation. For example, if the robot has to count aloud, as PAINT/1S2A(Sp) of section 1.1 does, then speech contexts would not be independent of the task situations.

This condition of there being a certain number of task independent rules is not a necessary condition. It can be relaxed in the following six ways;

1. Led actions may be transferred from any task situation to any other task situation without there being a group of task-independent contexts. There must be a certain amount of independence of contexts from the task, but there does not have to be a group of contexts that is independent of all task situations. Contexts may be independent of only some task situations, some being independent when others are not. For example, vision contexts might be used by a robot during one part of a task, and speech in another part of a task, but not both together. By "vision contexts" I mean contexts with only visual events in them. Similarly "speech contexts" are contexts with only speech events in

them. A teacher could use speech to transfer actions from one situation to another, as suggested above, while speech did not interfere with the task itself. When speech was being used by the robot, but not vision, the teacher might be able to use gestures to visually produce actions, in a similar way to the speech transfer of actions.

2. The condition of context independence can be relaxed further than indicated in 1. Contexts do not have to be independent of the task for the required action transfer to take place. For example, imagine that two contexts' predictions of an action are required for the action to be performed. There might be four types of context, C1, C2, C3 and C4. The relationships between the four types of context might be those shown in Figure IV-11. Contexts of type C1 are independent of all

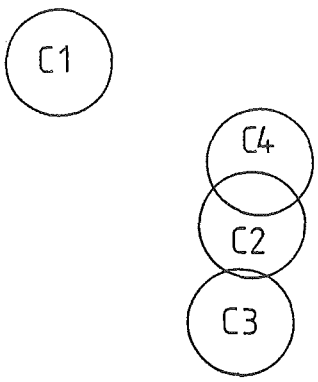


Figure IV-11 Transferring actions without using task-independent contexts.
Contexts of type C2 are dependent on the task conditions. If two contexts must predict an action for the action to be performed, then an action cannot be transferred to contexts of type C2 using contexts that are independent of task conditions. Only one type of context, type C1, is independent of the task conditions. However, since C1 and C4 are independent of C3, an action can be transferred to any C3 context. Thus an action can be transferred to any C2 context by (i) transfering it to the C3 contexts that occur with the C2 context, and then (ii) having these C3 contexts and the independent C1 predict the action.

other types of context. Type C3 is independent of type C4. Type C2 is slightly dependent on type C3 and dependent on type C4. Imagine that

contexts of type C2 are dependent on the task situations. Hence types C3 and C4 will be partly dependent on the task situations. An action can be generalized to C2 contexts using task dependent contexts. Suppose there are only three values that contexts of type C3 can have for all the possible task conditions. An action could be transferred to all three of the C3 values using the combination of C3-independent C1 and C4 contexts. Then an action could be transferred to a C2 context using the combination of C1 and the three C3 values.

3. Predictions of an action by each context of a particular MC might be formed by transferring the action to each context separately. Rather than transferring the action to all contexts at once, the action could be transferred to one context at a time. Then, when the contexts occurred together, they would all predict the action.

4. It may be sufficient for only some task situations to be able to have the led actions transferred to them. Other task situations may not need to have any actions transferred to them. The actions required might be able to be led in the task situation.

5. It may be possible to make some contexts independent of movement task situations by having "dummy" actions in the contexts. Dummy actions are actions that have no effect on the robot's body. They just go into contexts. The dummy actions would replace other actions that were dependent on the movement task situation. For example, an MCLS might have a context containing the last three actions performed by the robot. These three actions could be three arm movement actions; for example +10, +20 and +30. If the action +40 was led in this context then +40 would become associated with the context +10 +20 +30.

Whenever the robot performed the actions +10, +20 then +30, the action +40 would be predicted. This prediction of +40 depends on those last three actions being +10, +20 and +30.

Three dummy actions, for example, D1, D2 and D3, could have been performed after +10, +20 and +30, before the action +40 was led. Then the context D1 D2 D3 would be associated with +40. Now the action +40 will be predicted after any three arm actions, as long as the three dummy actions are put in the context after three arm actions are performed. The dummy actions enable the context to predict arm actions independently of the movement task situation. With dummy actions in it this context could be used in a similar way to the way speech contexts can be used to generalize actions. Dummy actions have been discussed by MacDonald (p.57. 1979. See also section A.3.2 in chapter VII), for the general learning of reflex actions, and used by Palfi (p.66. 1977b).

A form of action transfer that must be possible is the transfer of actions from leading situations to performance situations. For example, when a teacher leads an advanced future robot through a movement, and thus teaches an action, the robot will sense the teacher's touch, see the teacher, sense the relaxed state of its arm, sense certain forces on different parts of its arm, etc. All these conditions will be different when the teacher is not leading the robot. The robot must be able to perform the led actions regardless of this difference, or it would be quite useless. Leading would never enable it to perform actions!

IV.A.2.2 Correction-action generalization An MCLS may enable correction actions that are performed as a result of the teacher's speaking, to be performed without the teacher speaking. Suppose that a

robot has these production templates:

arm positon ---> arm actions (Arm rule)

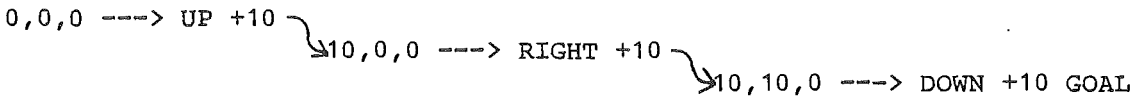
speech ---> arm correction actions (Speech rule)

touch ---> arm correction actions (Touch rule)

Suppose the teacher leads the robot through the sequence of movements,

up 10 cm right 10 cm down 10 cm,

from the position 0 cm, 0 cm, 0 cm, and rewards the robot at the end. Assume that the robot's actions are cartesian position changes. The internal representation stored in the Arm rule would be:



Now suppose a teacher leads the robot through a leftward movement of 5 cm from 20 cm, 20 cm, 20 cm, after saying "left". The robot stores the production "left" ---> LEFT +5 in the LTM of the Speech rule, and the production 20,20,20 ---> LEFT +5 in the LTM of the Arm rule.

Now say the robot is at 0,0,0 and it performs UP +10, and then RIGHT +10 at 10,0,0, but during the second movement it contacts an obstacle on its right. Suppose the teacher says "left". The robot adds LEFT +5 onto RIGHT +10, and so both performs and remembers the action RIGHT +5. The production 10,0,0 ---> RIGHT +5 would be put into LTM. Also the Touch production touch-right ---> LEFT +5 would be stored. Suppose the RIGHT +5 moves the arm to 10,5,0. Suppose further that there is a path or paths to the goal from there. The robot would follow a path to the goal.

Now, suppose that the robot performs the sequence again from 0,0,0. Suppose it avoids the obstacle it previously contacted during the rightward movement, by performing RIGHT +5 at 10,0,0, but contacts an obstacle on is right when performing a DOWN +10 action. The Touch

context "touch-right" will match the Touch production touch-right ---> LEFT +5, and LEFT +5 will be performed. Thus the DOWN +10 will be corrected to DOWN +10/LEFT +5. This corrected action may put the robot on a path to the goal.

The LEFT +5 was generalized from the situation with no touch, in which it was taught, to the situation with touch on the right when the teacher said "left". Then the MCLS performed LEFT +5 when touch-right occurred, even though the teacher did not say "left".

Whether or not this generalization of correction actions will be useful in a real robot has not been established.

IV.A.3 Advanced Robots : Leading and MCLSs

In section A.3.1 is discussed why the leading method implemented by an MCLS may be useful for teaching advanced future robots. In section A.3.2 are discussed the characteristics that may be needed in an implementation, by an MCLS, of leading in advanced future robots. Both these sections are necessarily somewhat speculative.

IV.A.3.1 Teaching using an MCLS and leading In addition to my main goal of designing robots that are more teachable and adaptable then existing robots, I have the long term hope of designing a robot that is something like humans. This is partly due to my interest in how humans learn and interact in the real world. I don't expect the robot to be made out of biological hardware, nor to mimic human behaviour precisely, but mainly to be like humans to interact with. This rather vague hope means that the robot will have to be able to be taught goals and sequences, for complicated algorithms. Even without my vague hope of robots being like humans to interact with, I expect robots of the future to be performing tasks that must be represented by

complex goals and sequences. So a system that provides an internal representation with both goals and sequences in it will be required.

For example, the task of riding a bicycle requires different movements, if the bicycle is small, from the movements required if the bicycle is large. Thus, one cannot always show a robot what to do by giving it a sequence of movements to perform. Something more is required if the robot is to accomplish the task in situations requiring different movement sequences. Giving the robot a goal of getting to a particular condition is not enough either, because the robot must go through the process of riding the bicycle.

As explained in section 1.1, an MCLS can provide an internal representation with both goals and sequences in it. Of course, I don't know what sort of MCLS is required for enabling a robot to ride a bicycle.

The rather vague hope mentioned above also means that the robots will not be able to have remote controllers or programming terminals, and will not be programmed in high level manipulation languages. Humans are not taught in these ways. Future robots would be able to employ the leading method, since humans can be physically taken through movements.

A more practical reason for not having remote controllers is that then they don't have to be designed and built. It could be very difficult and expensive to design a controller that enabled a teacher to show a robot, by remote control, how to ride a bicycle.

It has already been pointed out in chapters II and III that the leading method is a natural, easy to use method for (i) teaching a robot sequences of movements, and (ii) showing a robot individual movements for achieving goals. It may thus be preferred to programming

methods of teaching a robot to do complex movement tasks. I expect that programming methods will be extremely difficult for teaching robots how to do very complex tasks (see section 3.2 in chapter II).

The leading method is a more natural method, for humans to use to show robots or other humans movements, than either programming or guiding methods. Even with the sophisticated language which humans can use, it is sometimes easier to show a person a movement, than to explain it.

IV.A.3.2 Required properties of Leading in future Robots A future advanced led robot will need to (a) work out its own sequences of movements, and (b) overcome the incorrect information given to it during leading. Also, such a robot might be better not having a teacher-controlled training switch.

IV.A.3.2.1 Working out sequences of movements A teacher may not be able to lead or verbally guide a robot through the movements required for some manual tasks. If the teacher cannot give the robot the sequences of movements that are required, then the robot must work out the sequences of movements itself. It may be very difficult, if not impossible, to lead a robot through tasks that involve complicated movements and complex and difficult timing. The wrist movements we make when we play tennis require complex and difficult timing. Even someone who plays tennis very well is not aware of the subtly timed wrist movements he or she makes. In fact thinking about complex and subtly timed movements makes it much harder for us to do them (Blakeslee, 1980). Once we have led the robot through a manual task to the best of our ability perhaps all we can hope for is that the robot be able to perfect the task itself. It must work out the movements required. It could be difficult for us to use the verbal correcting

method because we would not know the movements required.

Also, there is unavoidable uncertainty about the environment (Fu, 1971). So a robot must work out its own sequences in the environmental conditions which the teacher did not teach the robot to cope with.

The inability of a teacher to know the movements for complex, subtly timed tasks differs from the physical inability of a teacher to lead tasks. A teacher may know the movements that are required for a task, even if he can't lead them. He can use sounds to verbally guide the robot through the movements.

An MCLS can (a) work out its own actions for achieving goals, and (b) transfer actions from the situations in which they were learned to other similar situations. These other situations might be unknown to the teacher.

IV.A.3.2.2 Correcting information acquired during leading A robot must be able to overcome the wrong information it obtains during leading. Firstly, the actions stored may not be the ones that are required for the task, as explained in chapter III, where a verbal correcting (VC) scheme was proposed for enabling a teacher to correct the actions. Secondly, the sensory information a robot obtains during leading may not be the same as the sensory information it obtains during its own performance of movements.

The differences in actions acquired and sensory information obtained are due to the teacher's influence on the interaction between the robot and its environment when he leads it through movements. For example, when a teacher pushes a robot along on a bicycle and makes steering movements he is affecting the balance of the bicycle. The teacher's effect is not present when the robot is riding the bicycle itself, so the balancing movements required then are different. The teacher may

be interacting with the robot in other ways while the robot is doing the task. For example the teacher might converse with or assist the robot. The effect will be different though, since the robot is now doing things itself. The achieving of different movements and the different sensory information are discussed below, in (a) and (b) respectively.

(a) Achieving different movements. Different movements are achieved by the robot itself because leading the robot by the hand does not solve the static and dynamic equations of its arm. This is because of the teacher's effect on the robot-environment interaction. When a teacher leads a robot's arm through a task he is giving the robot the joint trajectories it must achieve for the task. This does not tell the robot how to compensate for the opposing forces that its arm control system (ACS) does not compensate for, as explained in chapter III. For example, if the teacher leads the robot's arm through an upward movement then the robot will remember the joint trajectories for that movement. However, when the robot tries to repeat those trajectories the actual movement will depend on the inertial and gravitational effects of loads on the arm and on other dynamic effects. It there is a heavy weight on the robot's arm then the arm may sag. Thus the movement achieved may be different from the movement the teacher led the robot through. In the demonstration explained in chapter V a simple arm-robot with an MCLS overcomes the sagging effect of gravity. I have described how VC may enable led actions to be corrected. However, an MCLS may achieve the same effect without the teacher verbally correcting it. Its previous experience may enable it to do so. Section A.2.2 makes some comments on this. The arm-robot overcame the sagging using its previous experience.

(b) Different sensory information. The sensory information a robot obtains while being led through a movement may differ from the sensory information the robot obtains while doing the movement itself.

For example, imagine teaching a robot to ride a tricycle by putting the robot, which has arms and legs, onto a tricycle and pushing the tricycle along. The pedals would "lead" the robot's feet and legs through pedalling movements. The handlebars would lead the robot's hands and arms through steering movements as the teacher steered the tricycle. The robot would remember the pedalling and steering movements and perform them later by itself. However, some sensory information the robot obtains while riding the tricycle will differ from the sensory information it obtains while being led through the movements. For example, the pressure on the robot's feet would be less while it was being led along than while the robot was pedalling the tricycle along itself. The robot may need to use information about the pressure on the soles of its feet to tell if the tricycle is slowing down because of a rise in the ground or accelerating down a slope. Since it can generalize actions from one situation to another, an MCLS may enable a robot to perform a led task, even when some of the sensory information is different.

IV.A.3.2.3 No external training swtich A future advanced led robot may be better off without an external control for changing it from training mode to execution mode and back. The two main reasons for me wanting to design a robot whose mode control is not external are given below.

Firstly, an external mode changing button would allow tasks the robot had acquired to be destroyed, either deliberately or accidently. It would make no difference how well the task had been learned. One

would only need to push the button and lead the robot through "wrong" movements.

Secondly, an external button wouldn't be much use for teaching complicated tasks involving complex and subtle timing. I suggested above in section A.3.2.1 that one won't be able to teach a led robot to do tasks exactly, by leading it through them. If this is so then an external button won't be any use for difficult tasks. The teacher can only "do his best" in leading the robot through the task initially. Then the robot must perfect the task itself by obtaining and responding to information about the environment---and performing led actions to find out their true affects---as discussed in section 4.2 of chapter III. If the robot predicts an action, it should put itself into execution mode and perform the action. For example the wrist movements required for playing tennis are complicated and subtly timed.

The possibility of the teacher correcting his mistakes in leading the robot is made more difficult by not having the external button. The teacher could not lead the robot through the task again. However, as mentioned above, one doesn't want it to be too easy for tasks the robot does to be changed. Other ways for "correcting" teaching "mistakes" are discussed by Andreae (1977a).

There are two reasons why a robot with an external mode changing button could not fulfil my long term hope that a robot will behave and think like humans. Firstly, a robot with an external button would not have the independence we might expect of a humanlike robot. it would be unlikely to survive in a competitive environment of hostile robots. Humans survive in a competitive environment of sometimes-hostile humans. Secondly, humans don't have external buttons for putting them into and out of a passive training mode.

The example "arm-robot" in chapter V has no external training switch. The arm-robot has an MCLS in it.

CHAPTER V

A SIMPLE LED ROBOT WITH AN MCLS

This chapter reports demonstrations of leading implemented by an "arm-robot". The arm-robot comprises an MCLS and a real arm. The arm, shown in Figure V-1, is a single degree of freedom one; a metal bar. It has only one joint, which is rotational. The MCLS has two rules; ACTION rule and ANGLE rule. ACTION rule is a type (i) rule. Types of MCLS rule are described in section 2.1 of chapter IV. A type (i) rule has productions of the form <Context, action>. ANGLE rule is a type (iv) rule. It has productions of the form <Context, action, <Next context, prob(Next context)>, Goal, Val>. The production template of ANGLE rule is:

arm angle, arm velocity ---> arm action.

For example, 10 degrees, Stopped ---> +20 is an ANGLE production. The production template of ACTION rule is:

latest arm action, arm velocity ---> arm action.

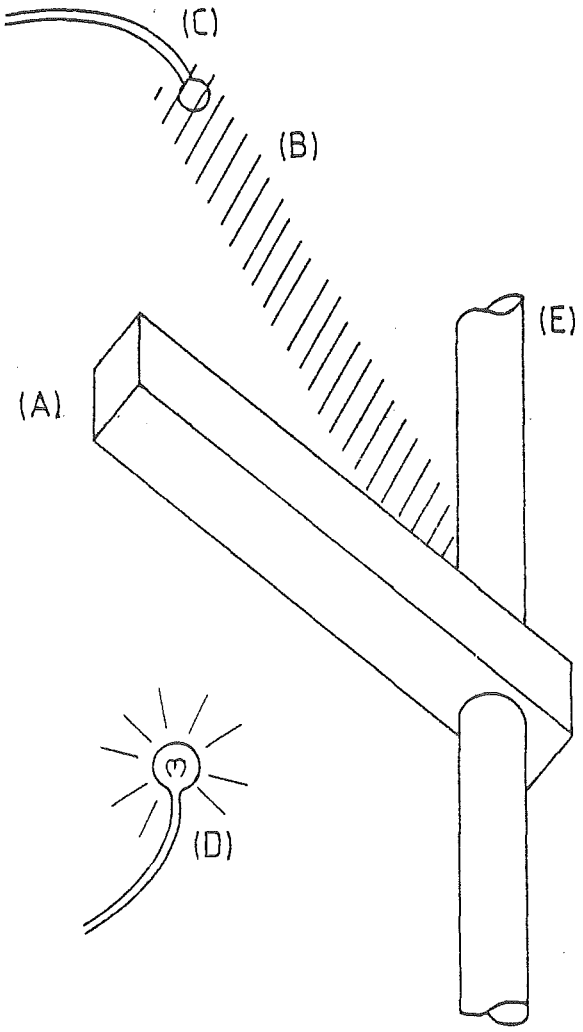
For example, +30, Up ---> -10 is an ACTION production. Both rules are choice rules since a context of either type may predict more than one action, as explained in section 2.1 of chapter IV.

There is goal-seeking only in ANGLE rule. An approximate form of leak-back, which is discussed in chapter VI, is employed in ANGLE rule. It enables an action, that is on a path to a goal in ANGLE rule's LTM, to be preferred to an action that is not on a path to a goal.

Briefly, the decision procedure is:

> If there is an action predicted by ANGLE rule
> that is on a path to a goal, then perform it.

Figure V-1 The single jointed lever arm of the arm-robot
The arm is a single bar which is attached at one end to a driving shaft. The arm is shown cutting a light beam. The task of the arm-robot is to lift and hold the arm in the light beam. The teacher teaches the robot this task by leading it through movements. For example, to teach the robot to perform a +40 action at zero degrees, the teacher quickly moves the arm up from zero degrees through a 40 degree movement to 40 degrees. Zero degrees is vertically downward.



(A) Lever arm
(B) Arm shadow
(C) Light sensor
(D) Light source
(E) Drive shaft

> If not then either perform an action that is predicted by both rules, if there is such an action, or else do not perform an action.

More detail is given in the appendix.

The task taught to the arm-robot is to lift its arm into a light beam, which has been set at a particular arm angle. This task is performed with a weight on the arm. The weight is not automatically counteracted by the arm control system (ACS) of the arm. The arm-robot MCLS learns to counteract the weight. It does this not by being verbally corrected---the arm-robot does not enable verbal corrections to be made---but by learning the actual effects of the actions it is led through. It learns to perform extra actions to compensate for the weight. The arm-robot learns to counteract the effect of gravity.

A short explanation of one teaching and execution session with the arm-robot, an "interaction", is given in section 1. Full explanations of this and three other interactions are given in the appendix. A simulated, idealized interaction is also reported there. Brief mention is made of another simulated interaction, and of nine real interactions, all made with an earlier version of the arm-robot. The arm-robot itself is described in detail in the appendix. The appendix is arranged in a number of sections, each giving more detail than the one before it.

Section 2 summarises the appendix discussion of the arm-robot and the interactions.

V.1 EXAMPLE INTERACTION WITH ARM-ROBOT

In this section I give an example of the simple arm-robot acquiring a task as a result of leading. The arm of the arm-robot is a metal lever, as shown in Figure V-1. The task is for the arm-robot to lift its arm and hold it in a light beam. The light beam is shown in Figure V-1. This "light beam task" must be performed with a weight on the arm. The weight causes the arm to sag. The arm-robot acquires the light beam task as a result of (a) being led through movements that bring its arm into the light beam, showing it the goal, (b) being led through other movements, which give it actions for counteracting the sagging caused by the weight, and (c) performing actions and discovering the movements they cause under the influence of the weight. Both the action sequence and the movement sequence performed by the arm-robot, in order to accomplish the task, are different from the led sequence of movements and actions. The robot selects its own actions for reaching the goal.

There is no verbal correcting in the arm-robot. Instead the teacher teaches only actual actions. The purpose of the demonstration given here is to show how an MCLS in a real physical robot can implement leading. It also shows how a goal-seeking (GS) system can implement leading. The ANGLE rule is equivalent to a GS system, since the contexts of the ANGLE rule are just Conds, the robot's conditions.

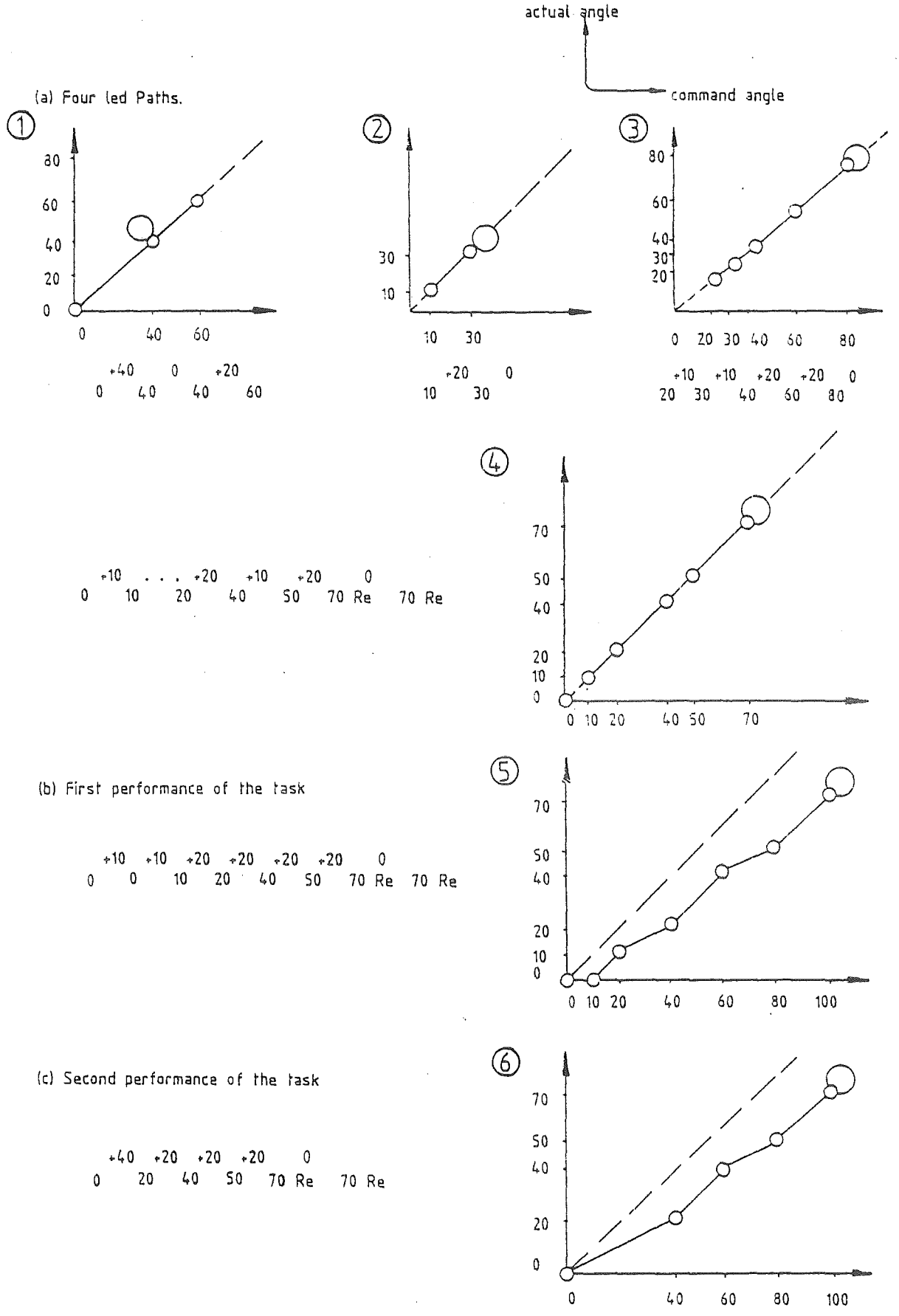
The arm-robot acquires the light beam task as a result of being led through the four sequences of actions and stimuli shown in Figure V-2(a). For example, to lead the arm-robot through Path 2 a teacher grasps the metal arm, and quickly moves it from 10 degrees to 30 degrees. He holds the arm there. Next the teacher leads the

Figure V-2 Acquiring the light beam task The simple arm-robot acquires the light beam task as a result of leading. The diagram shows the arm angle command signal, on the horizontal axis, versus the actual arm angle, on the vertical axis. All led actions have equal changes in the command and actual angles. The slopes of the lines showing led actions are unity. For example, in the first diagram in (a), a +40 action is led by the teacher moving the arm by 40 degrees. Some robot performed actions have greater changes in command angle than in actual angle. The slopes of the lines showing these robot performed actions are less than unity. For example, in (c), a +40 action moves the arm by only 20 degrees. This sagging is caused by a load on the arm. Unity slope is shown by a dashed line. The effect of each action is shown by a solid line segment with an open circle at each end. "Re" with 70 means that the arm cut the light beam at the arm angle 70 degrees. A hold or zero action is shown by an open circle with a circular solid line.

(a) The 4 led paths.

(b) First performance of the light beam task.

(c) Second performance of the light beam task.


actual angle
command angle
(a) Four led Paths.
①
80
60
40
20
0
0 40 60
+40 0 +20
0 40 40 60
②
30
10
10 30
+20 0
10 30
③
80
60
40
30
20
0 20 30 40 60 80
+10 +10 +20 +20 0
20 30 40 60 80
+10 . . . +20 +10 +20 0
0 10 20 40 50 70 Re 70 Re
④
70
50
40
20
10
0
0 10 20 40 50 70
(b) First performance of the task
+10 +10 +20 +20 +20 +20 0
0 0 10 20 40 50 70 Re 70 Re
⑤
70
50
40
20
10
0
0 10 20 40 60 80 100
(c) Second performance of the task
+40 +20 +20 +20 0
0 20 40 50 70 Re 70 Re
⑥
70
50
40
20
0
0 40 60 80 100

arm-robot through Path 3, and so on. [The complete sequence of movements and actions, including the not so important movements and actions performed between the Paths, is given in the appendix.] A goal is set automatically when the arm is in the light beam; at the end of path 4. A goal is set automatically whenever a light sensor senses the arm breaking the light beam. Arm velocity stimuli are not shown in the figure.

As a result of being led through these four sequences, in Figure V-2(a), the arm-robot produces the sequence of actions and stimuli shown as path number 5, in Figure V-2(b).

The robot seeks the goal at 70 degrees, through the transitions stored during the leading of Paths 1 through 4. The Vals of the productions on paths to the goal are higher than the Vals of other productions. The robot follows a path of high Val productions to the goal. The Vals are updated as goals and productions are stored. For example, as explained in detail in the appendix, the highest Val production with a context "0 degrees, Stopped" is the one predicting +10. So when stopped at 0 degrees, at the start of Path 5, the arm-robot MCLS selects +10. +10 is sent to the ACS. One may say that "reward leaks back" from the goal at 70 degrees, to 0 degrees, through the action +10. One can think of the high Val of the goal production as the concentration of a "chemical". The chemical leaks back through the paths of productions, its concentration decreasing as it leaks further back. The robot selects actions so as to follow the highest concentration of chemical. In fact, as explained in chapter VI, Vals may be calculated in an MCLS by a process that is analogous to this chemical leak back.

Later on, as a result of performing Path 5, in Figure V-2(b), the

arm-robot produces the sequence of actions and stimuli shown as Path 6 in Figure V-2(c).

This second robot-performed sequence is shorter than the first. The move from 0 to 70 degrees is accomplished in four individual movements, by four actions, on Path 6, while six actions are performed on Path 5. The light beam task is accomplished more quickly the second time it is performed by the arm-robot. The arm-robot is able to do this because of the information it obtains on the Path 5, the first performance of the task, about the effects of its actions on the weighted arm. For Path 6, the robot chooses the action +40 while at zero degrees, because reward leaked back through +40 from the context "40 degrees, Stopped" after the arm had reached 40 degrees on Path 5. Before Path 5 occurred, reward did not leak back to "0 degrees, Stopped" through the action +40. So +40 was not chosen at the start of Path 5. You can see in the Figure, on Path 5, that the arm passes through 40 degrees on its way to the goal at 70 degrees. Now, on none of Paths 1 through 4 does the robot establish a path to 70 degrees via 40 degrees, from 0 degrees. Path 1 goes from 0 to 40 degrees, but not to 70 degrees. Path 2 does not reach 40 degrees. Path 3 reaches 40 degrees but doesn't stop at 70 degrees. Path 4 goes to 70 degrees via 40 degrees, but does not start at zero degrees. The robot did not choose +40 at zero degrees on Path 5 because that +40, learned on Path 1, did not "connect" up to a path to 70 degrees. The "connection" is made on Path 5, but too late for the +40 to be chosen at 0 degrees.

The four led paths are taught in the order 1, 2, 3 and then 4. Path 4 is taught last so that the extra complexity of reward leaking back through the memory is not introduced into the robot's behaviour until near the end of teaching. The four paths are taught in quick

succession, each one following on from the last, after short term memory (STM) has cleared. The apparent simplicity of the teaching sequence is deceptive. This interaction has been selected, worked out by hand, and then verified on a computer simulation in order to demonstrate to the reader, in a short sequence, the learning that can result from leading.

The arm, the arm control system, and the MCLS of the arm-robot are explained in detail in the appendix. Briefly, the arm is a metal bar controlled by a second order servomotor system.

The arm-robot has no external switch for changing it from execution mode to leading mode and back again. This happens automatically. Briefly, the arm-robot automatically switches to leading mode, if no action is selected by the decision procedure, after a "time out" period of four seconds. The arm-robot automatically switches to execution mode if, during leading mode, the decision procedure selects an action. I shall call the leading mode also the "passive training" mode, since the arm is passive while in its leading mode. The execution mode will also be called the "active execution" mode, in order to emphasize that the robot arm is active during execution mode.

As well as the details of the arm-robot, thirteen interactions with the arm-robot are reported in the appendix.

V.2 DISCUSSION

Section 1 of chapter IV shows how an MCLS can perform sequences and achieve goals. The arm-robot has too simple an MCLS to perform sequences, so it is not an example of a sequence performing MCLS. The only sequence in the contexts is in the action, velocity contexts of ACTION rule. The arm-robot remembers the most recent action it performed.

The quantization of actions and stimuli in the arm-robot is crude in both time and magnitude. The arm-robot is a demonstration robot. A robot that performed complex manual tasks would need to have much finer quantization.

The appendix contains a detailed discussion of the arm-robot and the interactions. Five main aspects of the arm-robot are discussed. Firstly, I discuss the way the arm-robot determines the actions required for the task. This is compared to the requirements discussed in chapter II for determining actions, or motor commands, and for coping with a range of environmental situations. Briefly: (i) the arm-robot does not control the dynamic properties of the arm; (ii) the arm-robot can cope with the static effect of a weight; and (iii) the arm-robot's goal-seeking enabled it to cope with the situation of having a weight on its arm.

Secondly, three forms of improvement to the arm-robot are discussed: (i) improving the design of the single degree of freedom arm; (ii) increasing the number of degrees of freedom; and (iii) improving the MCLS in the arm-robot. For (i), a method is proposed whereby the feedback gains of the arm-robot controller could be adapted to unknown loads.

Thirdly, I discuss the way the arm robot is given a repertoire of movements by the leading method.

Fourthly, I discuss how leading shows the robot information about the environment, as well as movements to perform. Leading shows the robot stimuli that may be received from the environment, as well as actions to perform.

Fifthly, the asynchronous nature of the arm-robot MCLS is discussed. The arm-robot receives new angle and velocity stimuli whenever the arm angle changes, rather than receiving new stimuli at certain intervals. Other MCLSs have not had this asynchronous nature.

V.3 CONCLUSION

The arm-robot, comprising a simple real arm and a simple MCLS, demonstrates how an MCLS can implement leading. The teacher led the arm-robot through movements, and led it to the goal angle at the light beam. This leading enabled the arm-robot to counteract gravity, achieving the goal by lifting the weighted arm into the light beam.

APPENDIX FOR CHAPTER V

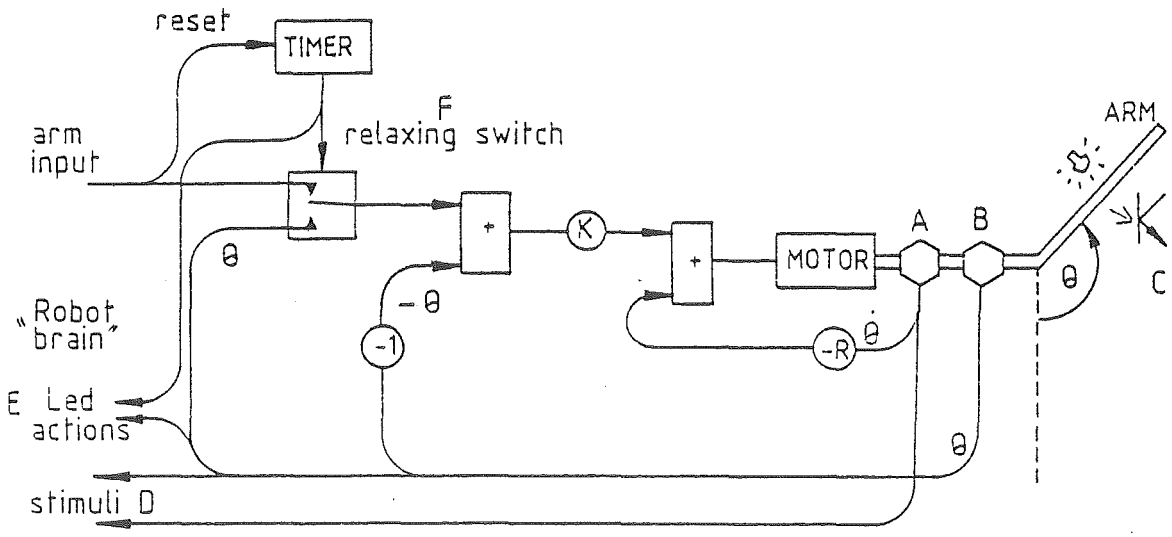
In section A.1 the example arm-robot is described. In sections A.2 and A.3 demonstrations, or interactions, of this arm-robot acquiring a simple task are reported. In section A.4 the arm-robot and the interactions are discussed. This discussion was summarized in section 2. Section A.5 gives circuit diagrams and program listings for the arm-robot. The arm-robot and interactions have been reported and discussed in MacDonald (1981; 1982a). There are two versions of the arm-robot. The earlier one is called the "arm-robot", and the later one the "leak-back arm-robot", in MacDonald (1981; 1982a). In this thesis the later one is called just the "arm-robot". The earlier one is called the "earlier arm-robot", and mentioned only briefly.

V.A.1 Description of the arm-robot: Example MCLS led robot

The arm-robot has been (a) simulated, and (b) implemented and tested with real physical hardware (MacDonald, 1981; 1982a). The simple robot arm and robot brain, which together make up the arm-robot, are described in sections A.1.1 and A.1.2, respectively.

V.A.1.1 Simple design for a robot arm

One of the simplest possible designs for a robot arm is a single degree of freedom, lever, arm with a torque motor actuator. The arm of the arm-robot is such a simple arm. It is a thin bar attached at one end to the geared down shaft of a torque motor. So it has one joint. The arm is shown in Figure V-1. Position error feedback must be provided so that the arm can be stable under the influence of gravity (Benati et al, 1980). Position error feedback provides stiffness. Figure V-3 shows a diagram of the simple arm with velocity feedback gain R and position error gain K. The velocity feedback ensures



A tachometer for velocity feedback

B potentiometer for position feedback

C the light beam is sensed by a phototransistor

D stimuli go to the MCLS "brain"

E Led actions are remembered when the arm is relaxed

F the relaxing switch is an internal mode button

Figure V-3 Simple design for a manipulator. Position feedback is required for stability against gravity. Velocity feedback is required for damping transients. The timer and relaxing switch control the training-execution mode changing.

stability and prevents excessively oscillatory behaviour of the arm.

The arm-robot has a leading, or passive training, mode as well as an active execution mode, so that it can be led through movements. The arm-robot's mode changing occurs automatically. There is an internal "button" rather than an external button. The robot arm automatically "relaxes" into training mode if the robot does nothing with the arm for more than a certain fixed time. A timer counts down from the fixed "relax" time of four seconds, towards zero. The timer is shown in Figure V-3. If the timer reaches zero then it operates the internal "button", or switch, putting the arm into a passive, relaxed, training mode by turning off the position error input to the motor. This is accomplished by switching off the input signal and switching in its place positive feedback of the arm angle. The positive arm angle signal cancels the negative arm angle feedback signal. There is no net position feedback, so the arm relaxes. It is not stiff. While in this relaxed training mode the robot "brain", an MCLS, remembers the movements the arm makes as the arm is led through movements. The MCLS brain is described in section A.1.2. If the MCLS brain sends a command to the arm for a movement then the timer is automatically reset to the relax time of four seconds. The timer immediately starts to count down. This causes the relaxing switch, or mode button, to switch back the position error signal. The arm "unrelaxes" into its execution mode. The motor then produces a torque which depends on the position error and the level of velocity feedback. The motor torque drives the arm towards the angle corresponding to the input signal. The electromechanical implementation of the arm and control system is described in section A.1.1.1 below.

V.A.1.1.1 Implementation of robot arm The single degree of

freedom arm is implemented with a modular servomotor system. The link of the arm is a metal bar made from folded sheet steel. The bar is 30 cm long and has a cross-section of 1.5 cm by 0.5 cm. The arm link is attached to a shaft which is driven through a series of reducing gears by a servomotor. The gear ratio is 18 to 1. The servomotor is a 1/50th horsepower, 6000 rpm, dc motor. Velocity feedback is provided by a tachometer on the motor shaft. Position feedback is provided by a rotary potentiometer on the arm link shaft. The modular servosystem implementation is given in section A.5.

The single degree of freedom arm system is controlled by a microcomputer via a conversion system. In the conversion system are a digital-to-analogue converter, an analogue-to-digital converter, a pair of two-position switches and a phototransistor. A circuit of the conversion system is given in section A.5. The digital-to-analogue converter converts an eight bit digital signal from the computer into an analogue voltage for the input signal to the arm system. The analogue-to-digital converter converts the analogue signals for the arm angle and arm velocity into eight bit digital signals for the computer. The computer uses one of the two-position analogue switches for selecting either the arm angle or velocity, for conversion by the analogue-to-digital converter.

The computer uses the other switch for switching positive arm angle feedback in place of the input signal. This switch is the internal "button" used for relaxing the arm into its training mode. The count-down timer required for operating the relax or training switch is implemented by the computer using a 25 ms hardware interrupt and an interrupt driven software count-down program. So the timer operates in real time, not computer "brain" time. There are two other timers

which are discussed in section A.1.2. They are implemented in the same way.

The conversion system provides a one bit signal which tells the computer whether a light beam is cut or not. The light beam is provided by a small incandescent bulb and is sensed by a phototransistor. The bulb does not actually provide a narrow beam. However, only when the arm blocks the line of sight path between the bulb and the phototransistor, does the phototransistor signal a "dark" state. So the effect of a light beam is produced. The task for the arm-robot is to hold its arm in this light beam.

The conversion system is controlled by the computer via a programmable three port device. Program listings are given in section A.5.

V.A.1.2 Robot "brain": remembering led movements and learning actively

The arm-robot MCLS has been specifically designed for the demonstration interactions reported in section A.2. The demonstrations are of the arm-robot acquiring the light beam task, and of the arm-robot performing the light beam task when there is a weight on its arm. The primary purpose of these demonstrations, and hence of the design of the arm-robot MCLS, is to give a real physical example of the way leading can be implemented with an MCLS. The demonstrations also show that an external mode changing button is not required for the leading method. However, the arm-robot MCLS is not to be construed as an MCLS for a robot that acquires and performs general movement tasks.

The templates for the two rules of the arm-robot MCLS are given at the start of this chapter. Actions are quantized into ten unit intervals. Angles are quantized into 10 degree intervals. A velocity stimulus is Up, Down, or Stopped (U, D or S).

Section A.1.2.1 explains goal-setting and goal-seeking in the arm-robot MCLS. Section A.1.2.2 describes the decision procedure. Section A.1.2.3 describes the interface between the MCLS and the arm. This interface is required because the MCLS deals with quantized actions and stimuli that are discrete in time. The arm, since it acts in the real world, deals with continuous values of arm angle, velocity and input signal, and acts continuously in time.

The main steps in the operation of the MCLS are listed in Figure V-4 with the numbers of the sections which explain them in detail. A diagram showing the operation of the MCLS is given in Figure V-5. Computer programs for the arm-robot MCLS and its interface to the arm are given by MacDonald (1982a) and in section A.5.

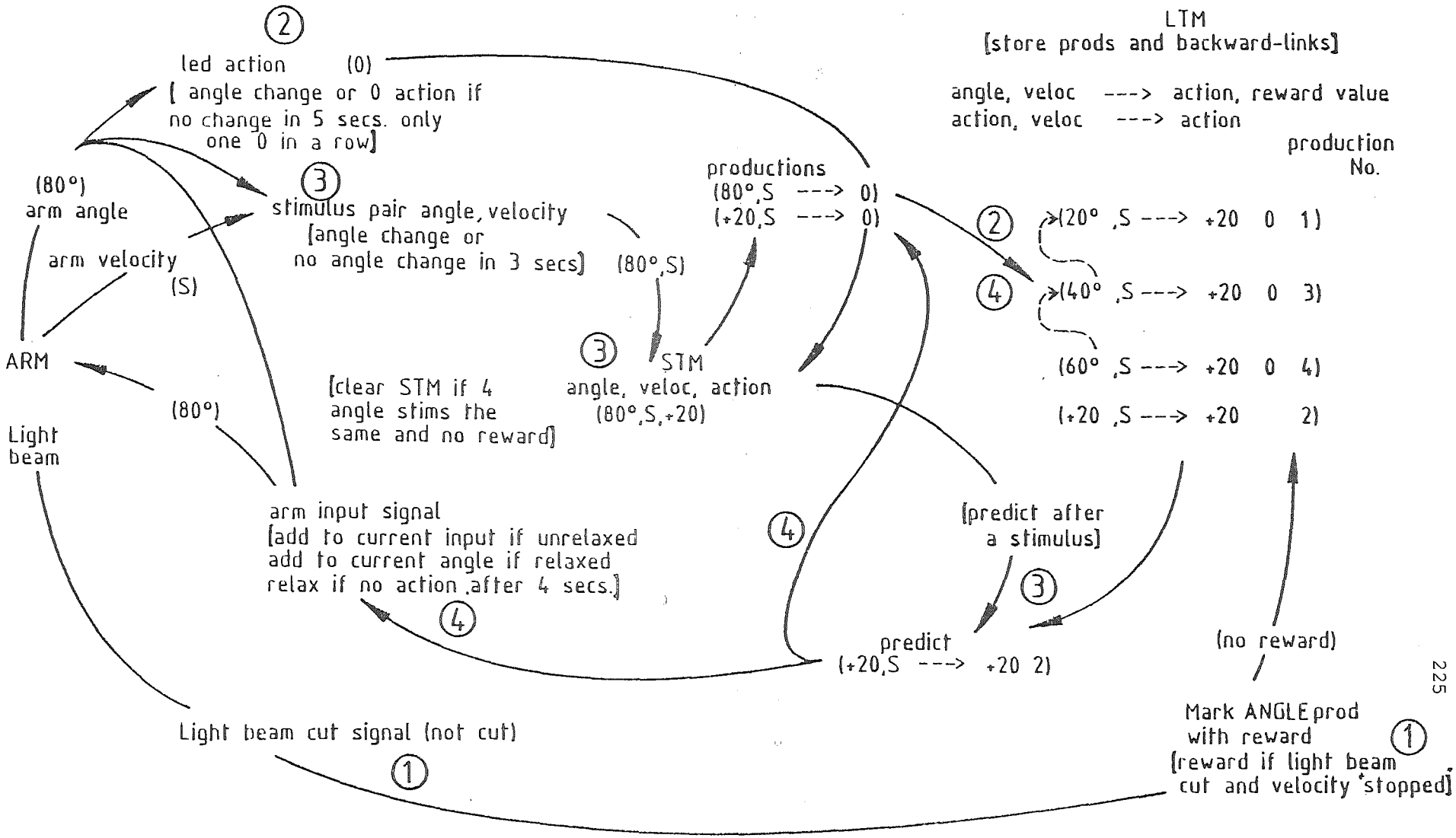
V.A.1.2.1 Reward A goal is stored in this way: ANGLE productions are stored with a reward code attached to them in LTM if the arm stopped in the light beam after the action was led or performed. For example, imagine that the ANGLE context is 60 degrees, Stopped when a +10 action is performed. If arm is stopped in the light beam following the occurrence of the +10 action, but before the occurrence of another action, then the production
60 degrees, Stopped ---> +10 will be marked as a "reward goal".

The transitions stored in the ANGLE rule are deterministic ones. There are no probabilities stored. The transitions point back from a context to the production that preceded it, rather than forward from a production to a context in the way shown in Figure III-6(a). This enables a very simplified form of goal-seeking to be implemented. Whenever the arm comes to rest in the light beam, a goal is set in the ANGLE rule, and reward is leaked back along the backward pointing

Figure V-4 The main steps in the operation of the arm-robot MCLS are listed with the numbers of the sections which explain them in detail. The circled numbers in Figure V-5 correspond to the main step numbers.

	STEPS	SECTIONS
1.	Mark the last ANGLE productions with reward if reward was obtained	A.1.2.1
2.	If the arm is relaxed then store a Led action if there is one	A.1.2.3(iii)
3.	Each time a stimulus pair arrives attempt to predict an action by comparing the contexts in STM with those in LTM	A.1.2.2 and A.1.2.3(ii)
4.	If there was an action predicted strongly enough then do and store the action chosen	A.1.2.2 and A.1.2.3(i)
5.	Go to 1.	

Figure V-5 Operation of the arm-robot MCLS "brain" with its interface to the arm. A detailed description of the MCLS is given in the text. A program for the MCLS is given in section A.5. Conditions and instructions are shown in square brackets. Examples are shown in parentheses. They correspond to the state of the arm-robot MCLS as the teacher is holding the arm up at 80 degrees and the robot is learning a 0, or hold, action. A +20 action is being predicted by production 2 but prediction by an ACTION production is not enough for an action to be performed. The decision procedure is explained in section A.1.2.2. Backward-links from angle contexts 40 degrees, S and 60 degrees, S to productions 1 and 3 respectively are shown. The circled numbers correspond to the main steps of the MCLS, which are listed in Figure V-4.


②
led action (0)
[angle change or 0 action if
no change in 5 secs. only
one 0 in a row]
③
stimulus pair angle, velocity
[angle change or
no angle change in 3 secs]
(80°)
arm angle
arm velocity
(S)
ARM
(80°)
Light
beam
(80°,S)
productions
(80°,S ---> 0)
(+20,S ---> 0)
③
STM
angle, veloc, action
(80°,S,+20)
[clear STM if 4
angle stims the
same and no reward]
arm input signal
[add to current input if unrelaxed
add to current angle if relaxed
relax if no action after 4 secs.]
④
④
predict
(+20,S ---> +20 2)
[predict after
a stimulus]
③
LTM
[store prods and backward-links]
angle, veloc ---> action, reward value
action, veloc ---> action
production
No.
②
(20° ,S ---> +20 0 1)
④
(40° ,S ---> +20 0 3)
(60° ,S ---> +20 0 4)
(+20 ,S ---> +20 2)
(no reward)
Mark ANGLE prod
with reward
[reward if light beam
cut and velocity stopped]
①
Light beam cut signal (not cut)
①

transitions. For example, when the angle context is 50 degrees, Stopped, and a +20 action is performed, which brings the arm into the light beam at 70 degrees, then the +20 in that context 50 degrees, Stopped becomes the goal. If there are pointers from 50 degrees, Stopped back to other productions, then these will be given a goal-value, or reward-value, that is less than the goal-value of 50 degrees, Stopped ---> +20. If these productions in turn have transitions that point back to other productions, then reward will be leaked back to these other productions, and so on. Figure V-6 depicts the formation of backward pointers, or backward links. Briefly, the decision procedure selects the nearest action to the goal in the ANGLE rule. This action will be an action with the highest reward value of all the actions that are predicted by the current ANGLE context. The backward links shown in Figure V-6(a) could be formed as a result of

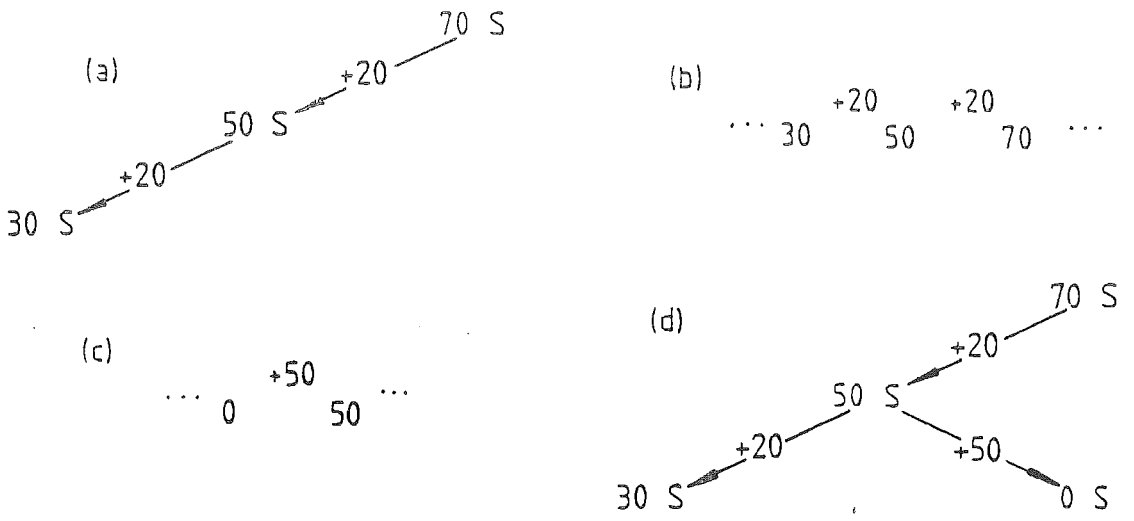


Figure V-6 Examples of leak-back. (a) A leak-back path from 70 degrees S to 30 degrees S. (b) The sequence of led actions and stimuli that resulted in the leak-back path shown in (a). (c) A +50 action is led at 0 degrees. (d) A leak-back path through 50 degrees S is added to (a), as a result of the leading in (c). In (a) through (d), "degrees" has been omitted from each arm angle. For example, "70" means "70 degrees".

the sequence shown in Figure V-6(b). Every time the robot obtains reward a simple version of a "leak-back" algorithm is executed. The leak-back algorithm in the arm-robot updates the Val of every production that has ever preceded the rewarded production's context. That is, reward is "leaked back" through the backward links. In Figure V-6(a) it can be seen that reward on the context 70 degrees, Stopped will leak back to 50 degrees, Stopped through the action +20, and then back to 30 degrees, Stopped through +20. The arm-robot could have been led through the sequence shown in Figure V-6(b) for this leak-back path to have been established. The amount of reward stored decreases by one unit per step as it leaks further away from the production which was marked as the reward goal.

In addition to any leak-back that occurs whenever reward is actually obtained, leak-back may occur whenever a new backward link is formed. Any reward on the current context is leaked back through the new backward link to the preceding production, to productions that preceded it, and so on. For example, if the arm-robot is led through an action +50 from 0 degrees, Stopped to 50 degrees, Stopped, as shown in Figure V-6(c), then a backward link from 50 degrees, Stopped through +50 to 0 degrees, Stopped will be formed, as shown in Figure V-6(d). Reward may be leaked along this path, influencing the selection of actions whenever the ANGLE context is 0 degrees, Stopped. Reward leaks back from reward-goal productions to all the productions that precede then in LTM. Rewards values affect the decision procedure in the way discussed below in 1 in section A.1.2.2.

V.A.1.2.2 Decision procedure

1. If ANGLE context predicts an action or actions marked with reward then do the predicted action which had the highest reward value given

to it most recently.

Otherwise,

2. If ANGLE and ACTION contexts both predict one or more actions then do the predicted action that was the first one stored.

Otherwise,

3. Do not perform an action.

In 2, there is no particular reason to take the first action ever stored. Any decision procedure must make some arbitrary choices (Newell, 1982). In 1, the choice of the most recent action is also arbitrary.

The robot can acquire productions while it is in its active execution mode, as well as during leading. Imagine that STM is 70 degrees, Stopped, 0. Imagine that the two productions 70 degrees, Stopped ---> 0 and 0, Stopped ---> -10 have previously been stored in LTM. 70 degrees, Stopped will be predicting 0, but 0, Stopped will be predicting -10. Now imagine that the action 0 is actually performed by the robot. The production 0, Stopped ---> 0 will be stored in LTM. The robot has acquired the production 0, Stopped ---> 0. Section A.2 demonstrates that actions can be performed as a result of prediction by an ANGLE context alone. Then the action is stored with the ACTION context as well, just like the 0 action being stored in the 0, Stopped ACTION context above. In the interactions that are explained in section A.2, the production 0, Stopped ---> 0 is acquired. It is acquired when the prediction of other actions by 0, Stopped is overcome by the prediction of 0 by 70 degrees, Stopped. The prediction of 0 "wins" over the others because the 0-predicting production was previously rewarded.

V.A.1.2.3 Interface between the arm and the MCLS The arm

operates in real time with continuous signals. The MCLS operates with events that occur discretely in time and are quantized in magnitude. The way discrete quantized events are transformed into and derived from continuous real time events is discussed in (i), (ii) and (iii) below.

Some hysteresis is built into the transformation of continuous real time signals into discrete quantized ones. This prevents the discrete quantized signals changing when there are small fluctuations of the continuous real time signals.

(i) Discretely timed quantized actions are transformed into continuous real time arm input signals. Actions cause relative changes in the input signal to the arm. The arm's actual input signal is held in a program variable and updated according to the following instructions for performing an action, for example the action +20:

1. If the arm is not relaxed then increase the absolute arm input signal by nx, where n is the number of tens of units in the action. For example, 2x is the increase in arm input signal for a +20 action. x is the change in input signal which causes a change of +10 degrees in the angle of the arm if the arm is not subject to external forces.
2. If the arm is relaxed then put the value n(y + x) into the absolute arm input signal program variable. y is the actual angle of the arm in tens of degrees. x and n are defined in 1. Thus if the arm is relaxed at some angle, say 20 degrees, and the action +20 degrees is performed, then the absolute arm input signal will be the one that corresponds to the arm being at 40 degrees.

(ii) Angle and velocity signals are transformed into stimuli. Angle stimuli are quantized at ten degree intervals. Velocity stimuli are

quantized into three levels. Angle stimuli and velocity stimuli are generated according to the following instructions:

1. If the arm angle crosses the threshold between one quantization level and another then send an angle stimulus, for the new angle, and a velocity stimulus for the velocity, to the MCLS. For example, suppose the arm angle changes from 22 degrees to 28 degrees and the arm is moving upward faster than the upward velocity threshold. The stimuli, 30 degrees, for the angle, and Up, for the velocity, would be sent to the MCLS.
2. If the arm angle doesn't change for more than three seconds then send the angle stimulus corresponding to the arm position to the MCLS, with the velocity stimulus. The STM is "cleared" if stimuli arrive three times in a row in this way, unless reward is being received. That is, if the same arm angle stimulus occurs four times in a row then STM is cleared, unless reward is being received. It will be seen in section A.2 that having the STM cleared of events is necessary to make the interactions given there possible.

(iii) While the arm is relaxed, led arm movements are transformed into actions to be stored. Actions are for relative changes in arm input signal, as stated in (i). When the arm is relaxed it can be led through movements, causing corresponding actions to be stored in productions in LTM, according to the following instructions:

1. Quantize the arm angle in 10 degree intervals.
2. If the quantized arm angle changes from its last recorded value by z lots of 10 degrees then send the action z0 to the MCLS for storing. z is an integer, other than zero. For example if the arm angle changes from 20 degrees to 38 degrees then the action +20 will be

sent to the MCLS for storing.

3. After an action has been sent to the MCLS for storing, if the arm angle does not change for five seconds, then send in the action 0 for storing. This allows a "holding" action, an action for holding the arm still, to be stored. A led "holding" action is not allowed to be stored immediately after another led holding action. A led holding action may be stored only straight after a different "led" action has been sent to the MCLS to be stored. For example, if the teacher leads the relaxed arm from 20 degrees to 38 degrees and holds it still, a +20 action will be stored, and then one 0, hold action.

It should be emphasised that the arm is relaxed throughout the execution of the three instructions, 1, 2 and 3, immediately above. As soon as the arm goes into its execution mode, the generation of led actions for storing ceases.

It is necessary for the arm-robot to have a 0, or hold, action. Its arm automatically relaxes after four seconds if the MCLS brain doesn't do any actions. The way the arm-robot holds its arm in one position is to keep doing 0 actions, preventing the relax timer from counting down to zero. Each time an action is performed the relax timer is reset to four seconds.

V.A.2 Arm-robot Interactions

In section A.2.1 two real interactions with the arm robot are described. In section A.2.2 both (a) two more real interactions with the real physical arm-robot, and (b) one interaction with a simulated version of the arm-robot, are described. In section A.3 interactions with an earlier version of the arm-robot, are briefly reported. These interactions have been reported before, in MacDonald (1981) and

MacDonald (1982a). The interaction number of the four arm-robot interactions is different here from in MacDonald (1982a). The present Interactions 1, 2, 3 and 4 correspond to Interactions 2, 4, 1 and 3 in MacDonald (1982a).

V.A.2.1 Two Interactions with the arm robot These first two interactions contain the important characteristics of all the interactions with the arm-robot, although all the interactions are different.

Figures V-2, V-7 and V-8 show two interactions with the arm-robot. In both of these interactions the arm-robot acquires the light beam task, and improves its performance of the task after accomplishing the task once.

These conventions apply to Figure V-7:

- the interactions are shown as a plot of stimuli versus "event time". The vertical scale is not real time but increases by one unit each time an action or stimulus occurs, the arm relaxes, STM is cleared, or reward is obtained

"-" means "Stopped" at the angle shown on the horizontal axis where the "-" is

">" means "moving Upward" at the angle on the horizontal axis

"<" means "moving Downward" at the angle on the horizontal axis

"R" means that the arm relaxed at the angle on the horizontal axis

"C" means that STM was cleared

"*" means the arm-robot was rewarded after the action which precedes the "*"

- actions are shown as single digit numbers for tens of action units. There are separate columns for "Led" and robot-performed, "Ac", actions

- reward values are shown with each performed action. A reward value of 255 indicates that the action has previously obtained reward for the arm-robot. The action put or held the arm in the light beam. Reward decreases by one unit for each step it leaks back from a rewarded action. So an action that is three steps, or actions, away from a rewarded action has a reward value of 252. Actions that are performed because of combined predictions by ANGLE and ACTION productions, but with no reward leaking to them, are shown as having a reward value of 2.

The arm drive has hysteresis, friction and stiction. Also, the torque produced by the motor while it is stationary varies considerably with the relative position of the rotor and stator of the motor.

In Interaction 1, the teacher intended to lead a specific sequence, which he had previously worked out. In Interaction 2 the teacher had no specific sequence in mind. All he intended to do was (i) raise the arm to 100 degrees, to show the robot how to make the input signal to the arm 100, the signal required for reaching 70 degrees with a weight, and (ii) raise the arm into the light beam, to set the goal. I did all the teaching myself, but write "the teacher" in order to emphasize the teaching and the arm-robot, rather than the teacher.

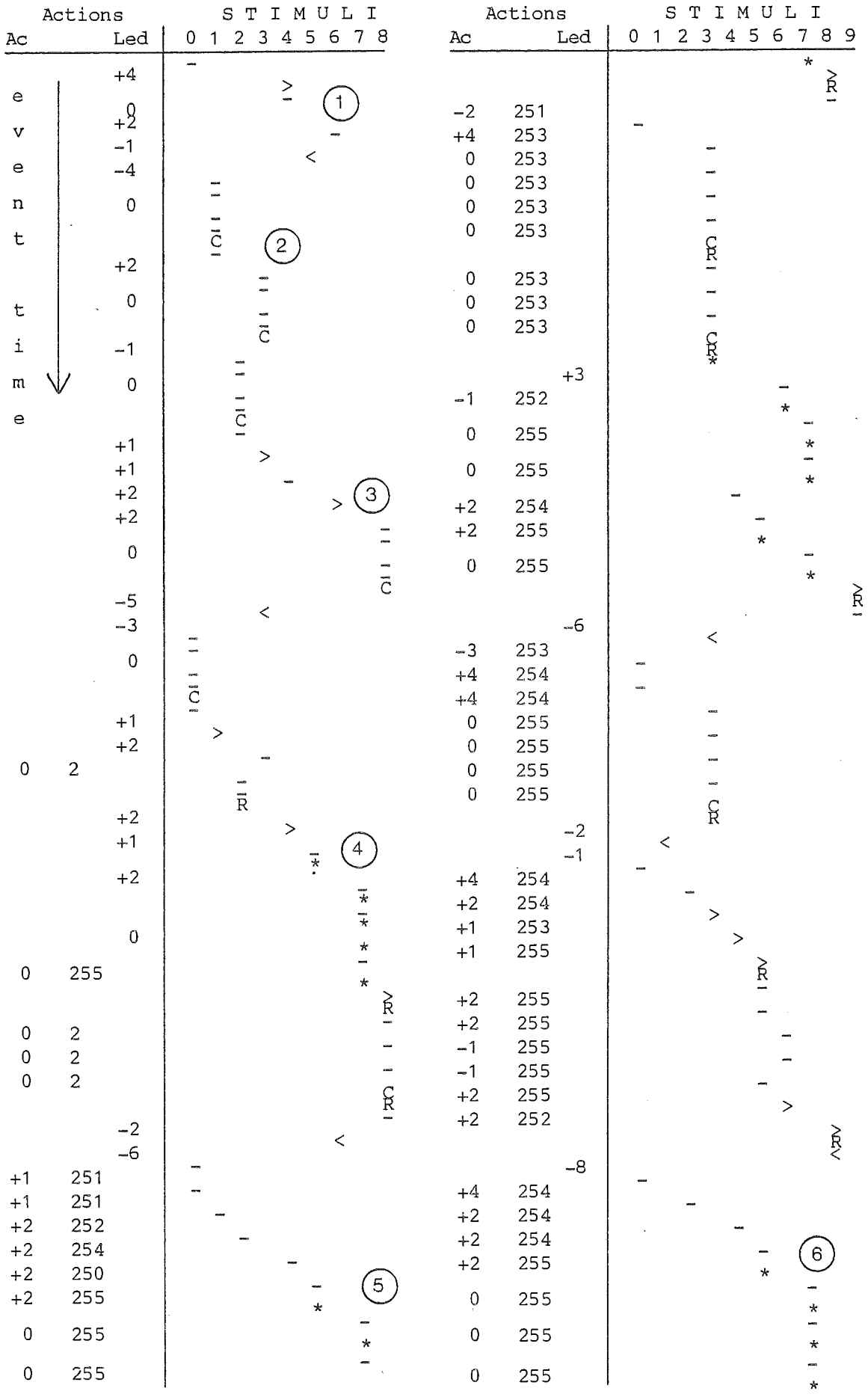
INTERACTION 1

The five main points of the first interaction, Interaction 1, are summarized below. Interaction 1 was briefly explained in section 1.

1. The teacher leads the arm-robot through four paths of movement, as show in Figures V-2 and V-7(a). The robot performed the light beam task twice, on Paths 5 and 6. These six Paths are shown in Figure V-2. The entire interaction is shown in Figure V-7(a).

Figure V-7 (caption on next page)

(a) INTERACTION 1



(b) INTERACTION 2

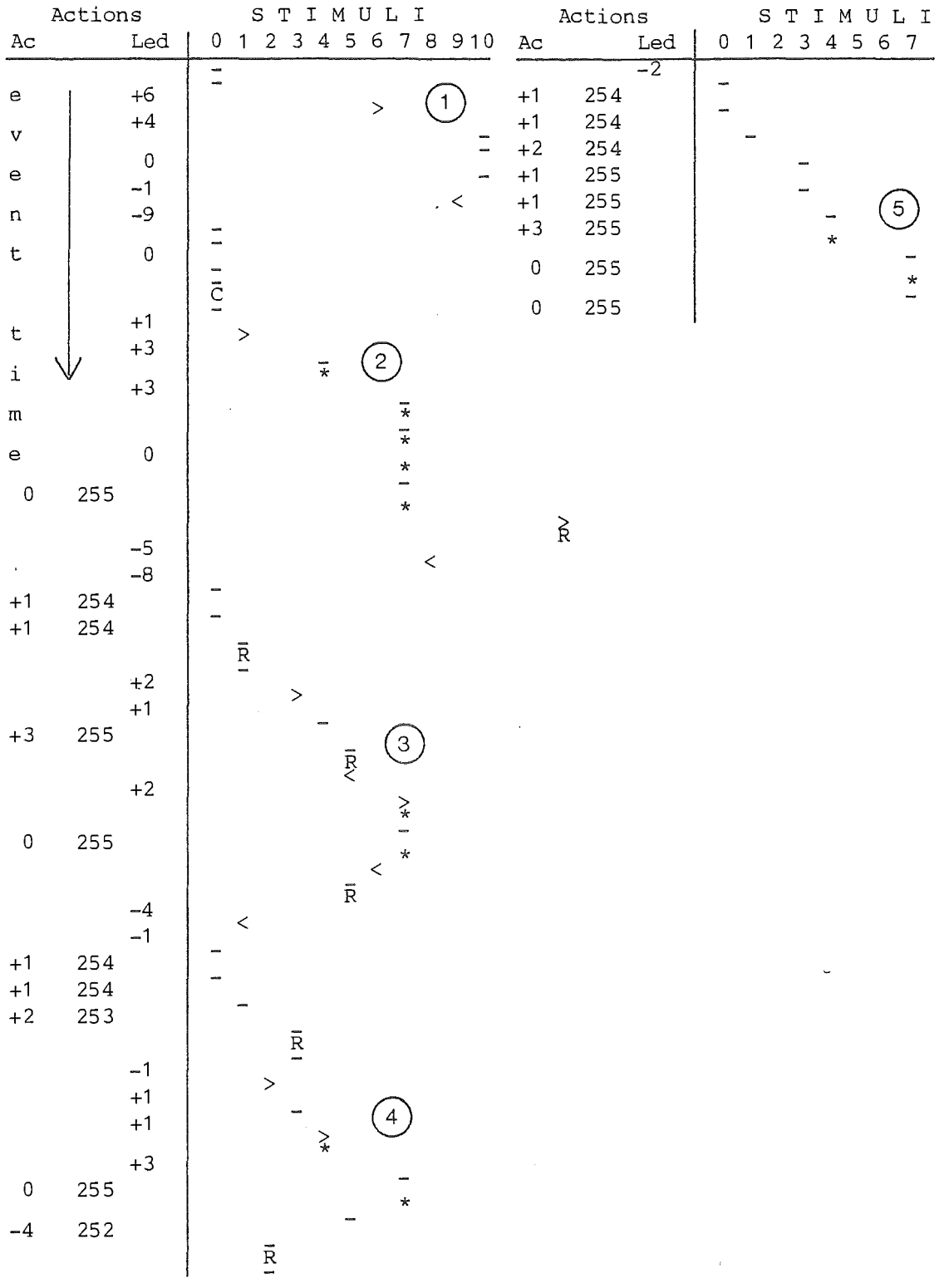
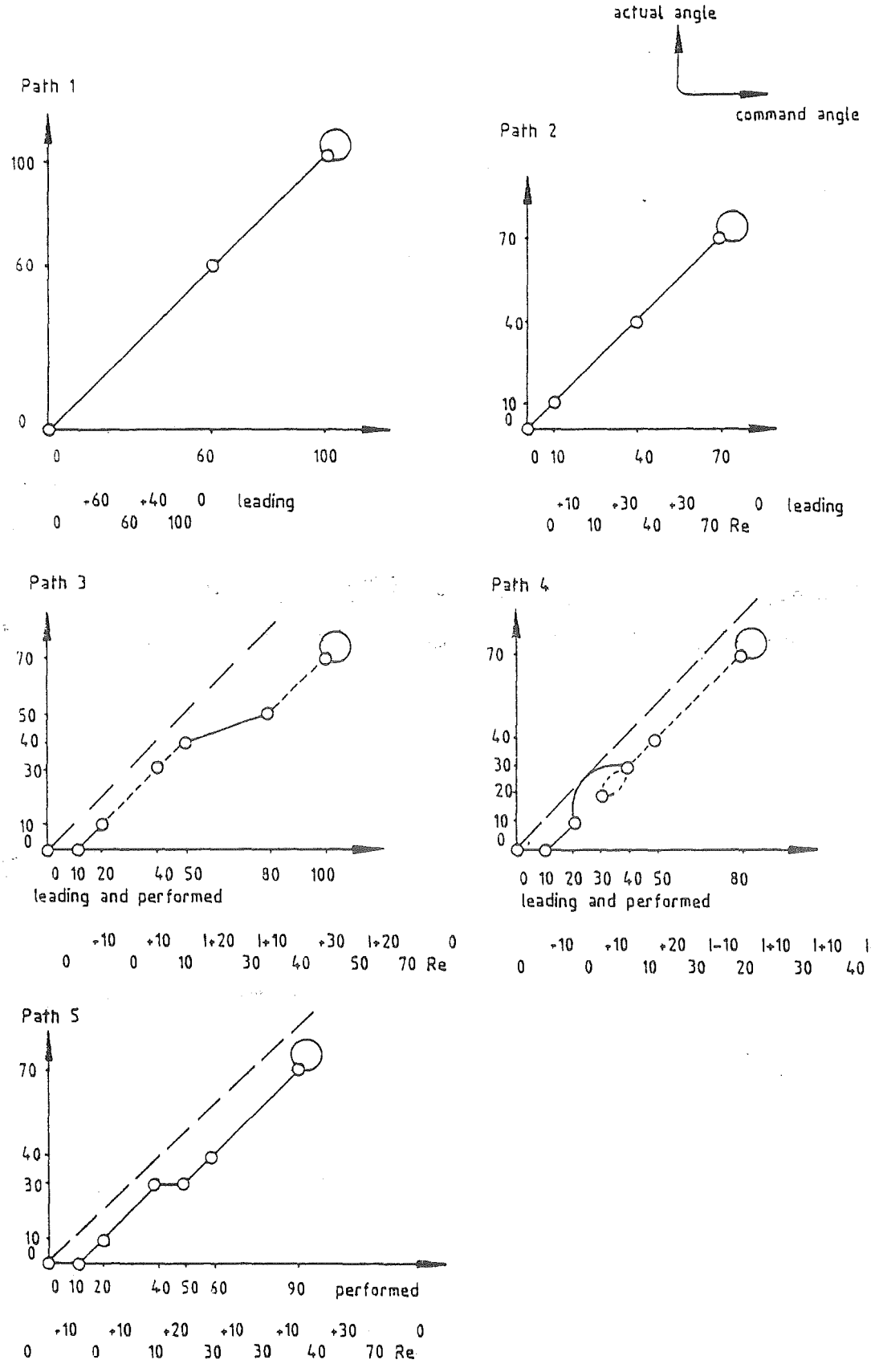


Figure V-7 Two interactions with the arm-robot The arm-robot improves its performance of the light beam task by trying to do the light beam task. The paths of Figures V-2 and V-8 are shown by circled numbers. The diagrams are explained in the text.

Figure V-8 Main Paths for Interaction 2 The diagram shows the arm angle command signal, on the horizontal axis, versus the actual arm angle, on the vertical axis. All led actions have equal changes in the command and actual angles. The slopes of the lines showing led actions are unity. For example, on Path 1, a +40 action is led by the teacher moving the arm by 40 degrees. Some robot performed actions have greater changes in command angle than in actual angle. The slopes of the lines showing these robot performed actions are less than unity. For example, on Path 3, a +30 action moves the arm by only 10 degrees. This sagging is caused by a load on the arm. Unity slope is shown by a dashed line. the effect of each action is shown by a line segment with an open circle at each end. The dotted lines, in the Paths with both led and performed actions, are for the led actions. "|" before the action means it was a led one. "Re" with 70 means that the arm cut the light beam at the arm angle 70 degrees. A hold or zero action is shown by an open circle with a circular solid line.


actual angle
command angle
Path 1
100
60
0
0
60
100
+60 +40 0 leading
0 60 100
Path 2
70
40
10
0
0 10 40 70
+10 +30 +30 0 leading
0 10 40 70 Re
Path 3
70
50
40
30
10
0
0 10 20 40 50 80 100
leading and performed
+10 +10 I+20 I+10 +30 I+20 0
0 0 10 30 40 50 70 Re
Path 4
70
40
30
20
10
0
0 10 20 30 40 50 80
leading and performed
+10 +10 +20 I-10 I+10 I+10 I+30 0
0 0 10 30 20 30 40 70 Re
Path 5
70
40
30
10
0
0 10 20 40 50 60 90 performed
+10 +10 +20 +10 +10 +30 0
0 0 10 30 30 40 70 Re

2. On Path 4 the teacher leads the robot's arm into the light beam, causing a reward goal to be set up for that arm position. That is, the teacher leads the arm-robot through the light beam task. He leads the arm into the light beam. However, the sequence of actions that the teacher led the robot through was not a sequence that the robot could use to lift its arm into the light beam from zero degrees. The weight causes the arm to sag.

3. Actions led on Paths 1, 2 and 3 are combined with Path 4, to produce first Path 5, then later Path 6. The task is performed despite the arm's sagging. Both the +20 action led from 10 degrees on Path 2 and the +20 action led from 40 degrees on Paths 1 and 3 are incorporated into the performance of the task on Path 5. Both the +40 action led from zero degrees on Path 1 and the +20 action led from 40 degrees on Paths 1 and 3 are incorporated into the performance of the task on Path 6.

4. After the leading of the four Paths, reward leaks back to zero degrees through a +10 action. Just how this occurs is explained below. The leak-back to 0 degrees through +10 causes the arm-robot to perform a +10 action, then another +10 action and so on, into the led Path 4. However the arm does not behave as it did during the leading, because the weight causes it to sag. Nevertheless, because of the leading that the teacher did on the first three Paths, the arm-robot has actions it can perform at the positions the arm sags to. It is able to reach the light beam along Path 5.

5. The teacher lifts the arm out of the light beam. The arm relaxes. The teacher releases the arm. The arm drops to zero degrees again.

Now, new backward-links were formed during the execution of Path 5, because of the sagging behaviour of the arm. Thus reward leaks-back to zero degrees more strongly through a +40 action than the +10 action. The robot performs +40, and so on, executing Path 6. Path 6 is shorter than Path 5, in that fewer actions are performed to get from 0 degrees to 70 degrees. The arm-robot improved its performance of the light beam task by performing the light beam task. It learned the effects of its actions.

I will now give a detailed explanation of Interaction 1. The teacher starts by jerking the arm up from zero degrees, which is shown by "-" in the "0" column of Figure V-7(a), to 40 degrees, which is shown by ">" and "-" in the "4" column for 40 degrees. The ">" means that arm was moving Upwards when the arm reached 40 degrees. A +40 action is learned by the arm-robot, as shown by "+4" in the "Led" column in Figure V-7(a). The +40 action is learned in the ANGLE context 0 degrees S. Stimuli and led actions come into the arm-robot's MCLS as fast as they can be handled by the MCLS. However the teacher can move the arm fast enough for some 10 degrees changes in angle to not be "seen" by the MCLS. So when the teacher jerks the arm by +40 degrees, a single, +40, action is learned.

A hold action is learned at 40 degrees. This is because the arm stopped there for more than five seconds after acquiring the +40 action. The learning of hold actions is explained in section A.1.2.3 (iii). The teacher jerks the arm up to 60 degrees. This movement from 40 degrees to 60 degrees causes a +20 action to be learned in the ANGLE context 40 degrees S and in the ACTION context +40 S.

The teacher lets the arm drop down to 10 degrees with a -10 action and a -40 action. Now he holds the arm stationary until STM clears, as shown by the "C" after four 10 degree stimuli. The teacher goes on to lead the robot through Paths 2 and 3.

The teacher then starts leading the robot into Path 4. At 30 degrees the robot does a 0 action. The 0 action is performed because the arm-robot learned to perform a 0 action at 30 degrees, at the end of Path 2. The arm unrelaxes when the 0 action is performed. The teacher, not expecting an action to be performed, but feeling the arm become stiff, lets it go. The arm sags under gravity, to twenty degrees. The teacher keeps his hand just under the arm. When the arm relaxes the teacher leads it on through Path 4. Path 4 brings the arm up to the light beam reward angle of 70 degrees, as shown by the first "*" in Figure V-7(a). The teacher has led the arm into the light beam, obtaining reward for the arm-robot and causing the production "50 degrees, Stopped ---> +20" to be set as a goal.

The teacher holds the arm in the light beam at 70 degrees, teaching the arm-robot to do a 0 action at 70 degrees. The production "70 degrees, Stopped ---> 0" is stored as a goal production. Having learned to do a 0 action at 70 degrees, the arm-robot does a 0 action there. The teacher then lifts the arm up to 80 degrees. STM clears, the arm relaxes and the teacher jerks the arm down to 0 degrees. Reward leaking back along Path 4 in the LTM of the MCLS causes the MCLS to do a +10 action from 0 degrees. The +10 action does not lift the arm up to 10 degrees because of both the weight of the load and stiction in the arm drive. A second +10 lifts the arm up to 10 degrees. A +20 action is performed, as learned on Path 2. This +20 action brings the arm to only 20 degrees because of the weight of the

load. A +20 action is performed, as learned on Path 4. The arm moves to 40 degrees. Another +20 action, led on Paths 1 and 3, is performed. The arm moves to only 50 degrees, because of the weight. A further +20 is performed, bringing the arm to 70 degrees. Reward is obtained. The robot has reached its goal.

The teacher lifts the arm to 80 degrees. The arm relaxes and drops to zero degrees. A -20 action is performed as the arm drops. The -20 action unrelaxes the arm at an angle, say x degrees, that is between 80 degrees and 0 degrees. Thus the +40 action that follows is added to an arm input signal that is x minus 20. The +40 action brings the arm to 30 degrees, instead of 20 degrees. So the arm probably unrelaxed at about 30 degrees, since 30 minus 20 plus 40 is 50. As shown at the end of this interaction, a +40 action performed from 0 degrees brings the arm to 20 degrees and another +20 takes it to 40 degrees. Once at 30 degrees the arm-robot goes on performing 0 actions. This is because the arm was dropped to 20 degrees after the arm-robot learned its first 0 action at 30 degrees. Reward leaked back to 20 degrees S and through 0 to 30 degrees S. The teacher leads the arm through a +30 action when, for a second time, STM clears and the arm relaxes at 30 degrees. He holds the arm at 70 degrees then releases it. When the arm reaches 70 degrees again he lifts it to 90 degrees. Then after the arm relaxes at 90 degrees another sequence brings the arm to 30 degrees doing 0 actions. Again the arm relaxes. A +40 action is performed from 0 degrees. However the arm relaxes on the way up, and ends up at 80 degrees. The arm relaxes again and falls to 0 degrees. Finally the arm-robot performs Path 6, the sequence +40, +20, +20, +20 to reach 70 degrees.

INTERACTION 2

The second interaction is quite different from the first, and in fact from the other interactions reported by MacDonald (1981; 1982a), and explained here. Those other interactions are similar to, but not the same as, Interaction 1.

The leading and teaching of the light beam task occur together in Interaction 2. Firstly the task is set up as a goal. Then the robot "tries" to perform the task. The robot arm automatically relaxes whenever the arm-robot does not perform an action. Each time this relaxing occurs, the teacher leads the arm-robot through an action. Thus, next time the arm-robot is in the same situation it performs that action, rather than relaxing. Eventually the arm-robot is able to perform the whole task itself. The four important Paths in Interaction 2 are shown in Figure V-8. The entire interaction is shown in Figure V-7(b).

First the teacher lifts the arm straight up to 100 degrees, causing a +60 action and a +40 action to be learned, with 60 degrees, U in between. This is Path 1. The teacher lets the arm drop. He then lifts the arm to 40 degrees, causing a +10 and a +30 action to be learned, and on to 70 degrees with a +30 action. He holds the arm in the light beams at 70 degrees. This is Path 2. The arm-robot learns a 0 action, and then performs a 0 action. The teacher lifts the arm to 120 degrees, the arm relaxes and drops to 0 degrees. The arm-robot starts performing actions to lift its arm up. Each time the arm-robot does not perform an action, resulting in the arm's relaxing, the teacher holds the arm to prevent it from dropping. He then leads it through an action from the position where it relaxed. For example the arm relaxes when the robot does a +10 action and receives the stimulus

pair 10 degrees S, which it has never had before. The teacher leads the arm through a +20 action. This is on Path 3. Next time the arm-robot reaches 10 degrees S it does the +20 action itself, on Path 4, and so on. Eventually the arm-robot is able to lift its arm into the light beam by itself.

These two interactions show the arm-robot improving its performance of the light beam task by "trying" to perform the light beam task.

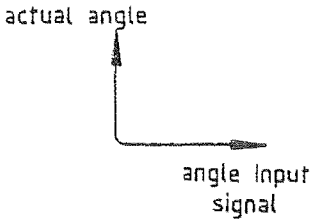
The task that was acquired and performed by the arm-robot was a task which required the robot both to seek a goal and to compensate for external forces.

A simulated interaction and two more real interactions are explained below, in section A.2.2.

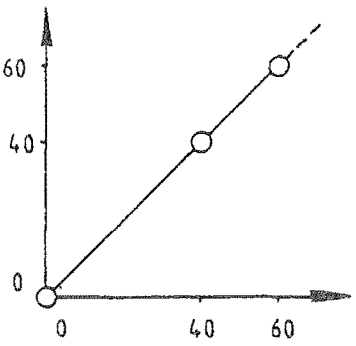
V.A.2.2 One simulated and two real interactions Figure V-9 shows paths of actions and stimuli which a simulated version of the arm-robot was either led through---Paths 1 to 4---or followed by itself---Paths 5 and 6. This simulated interaction is an ideal interaction which did not actually happen with the arm-robot. The simulated interaction ignores friction, stiction and other aspects of the arm-robot that are not fundamentally important to my demonstrating, in an idealized fashion, the arm-robot's learning and improving of the light beam task. Two real interactions with the actual arm-robot are also discussed below. Two other real interactions were discussed above in section A.2.1. The four actual interactions with the real arm-robot show that the simulated interaction captures the essential aspects of the task acquisition and improvement.

The important parts of the hypothetical or stylized, simulated interaction are the six paths of actions and stimuli shown in Figure V-9. The first four are listed below. They are led sequences.

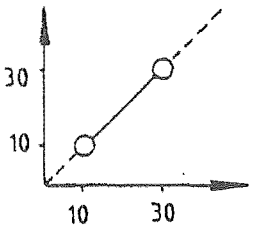
Figure V-9 The six Paths of actions and stimuli. Paths 1, 2, 3 and 4 are led by the teacher. Paths 5 and 6 are performed by the simulated arm-robot. The diagrams show the arm angle input signal, on the horizontal axis, versus the actual arm angle, on the vertical axis. As shown in Paths 1, 2, 3 and 4, all led actions are along path segments which have equal changes in vertical and horizontal directions. The slope of each led path segment is unity. For example, a +20 action is led by the teacher moving the arm by 20 degrees. Some robot performed actions are along path segments which have greater horizontal components than vertical components. The slope is less than unity. For example the +40 action performed at 0 degrees in Path 6 moves the arm by only 30 degrees. The horizontal change for the +40 action is 40. The vertical change is only 30 degrees. The sagging is caused by the weight on the arm. In each diagram, unity slope is shown by a dotted line. The effect of each action is shown by a solid line segment with an open circle at each end. A hold or 0 action is shown by an open circle with a circular solid line.



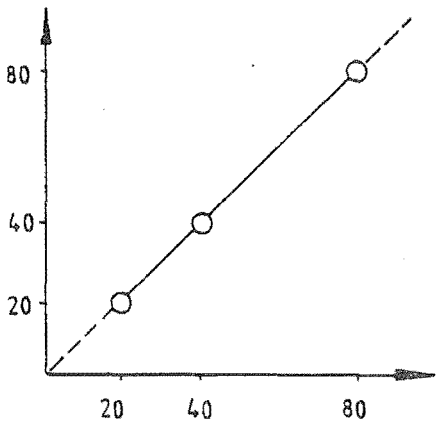
Path 1



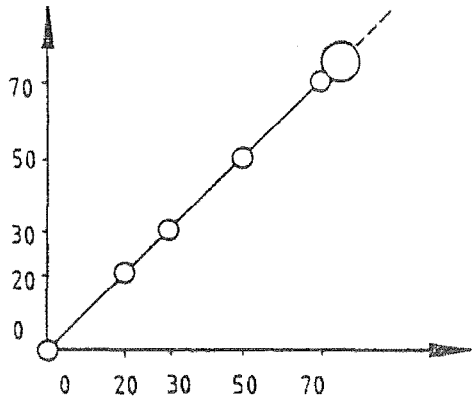
Path 2



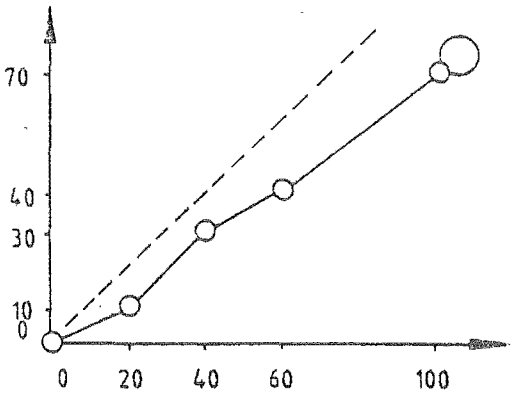
Path 3



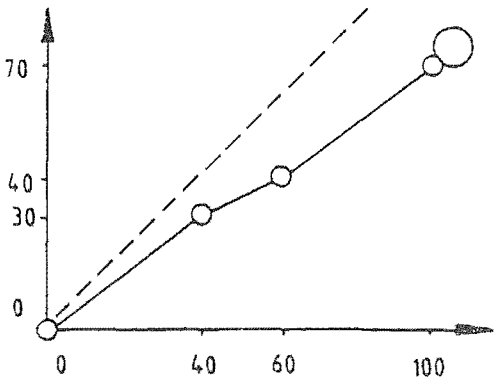
Path 4



Path 5



Path 6



```
Path 1      +40      +20
        0 deg  40 deg  60 deg

Path 2       +20
        10 deg  30 deg

Path 3       +20      +40
        20 deg  40 deg  80 deg

Path 4      +20      +10      +20      +20          0           0 ...
        0 deg  20 deg  30 deg  50 deg  70 deg Re  70 deg Re
```

For example, in order to lead the arm-robot along Path 1 the teacher "jerks" the arm from zero degrees---which is vertically downward---up to 40 degrees, stops briefly, and then "jerks" the arm up to 60 degrees. The leading process is simulated in the simulated interactions. The teacher need only push some keys on a computer keyboard to lead the simulated "arm" through movements. The teacher now lets the arm drop to 10 degrees and waits for the contexts in STM to clear. The clearing of STM is specified in section A.1.2.3 (ii). Then Path 2 can be led, and so on. Reward is obtained at the end of Path 4. This reward leaks back through the MCLS's memory, as shown in Figure V-10(a). The dropping down movements are not as important as the Paths.

Once the four Paths have been led the arm is let drop to 0 degrees. The arm-robot will start off along Path 4, because reward leaks back to 0 deg through the action +20, as shown in Figure V-10(a). However, the arm has a weight on it. This simulated arm-plus-weight behaves as shown below,

requested input angle 0 1 2 3 4 5 6 7 8 9 10

actual angle achieved 0 1 1 2 3 3 4 5 5 6 7

The following sequence, which is Path 5 in Figure V-9, would result.

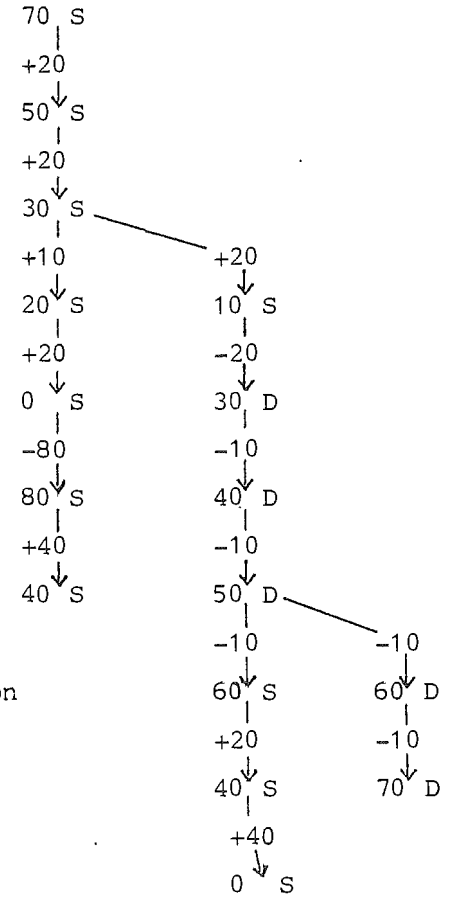
```
Path 5      +20      +20      +20      +40          0           0
        0 deg  10 deg  30 deg  40 deg  70 deg Re  70 deg Re
```

Note that the weight caused the arm to only reach 40 degrees after the +20 action performed at 30 degrees. This caused the arm-robot to

Figure V-10

(a)

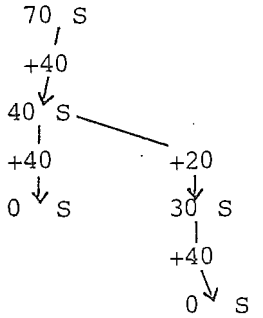
Leak-back tree



Reward leak-back from the context 70 S to 0 S after the robot has been led through the first four Paths. The robot does a +20 action from 0 S because the reward leaks back with the least decrement to 0 S via the +20 action. The +20 action is on the shortest path in memory from 0 S to 70 S. The +20 action brings the arm to 10 S, where another +20 action is performed.
The links and productions acquired during the interaction in Figure are shown in Figure V-12.
Note that the degrees symbols "°" have been omitted from the Figure.

(b)

Leak-back tree



Leak-back from context 70 S to 0 S after Path 5 has been performed by the robot. Only the links that cause actions to be performed on Path 6 are shown. The links shown in Figure (a) are still present. Note that the degrees symbols "°" have been omitted from the Figure.

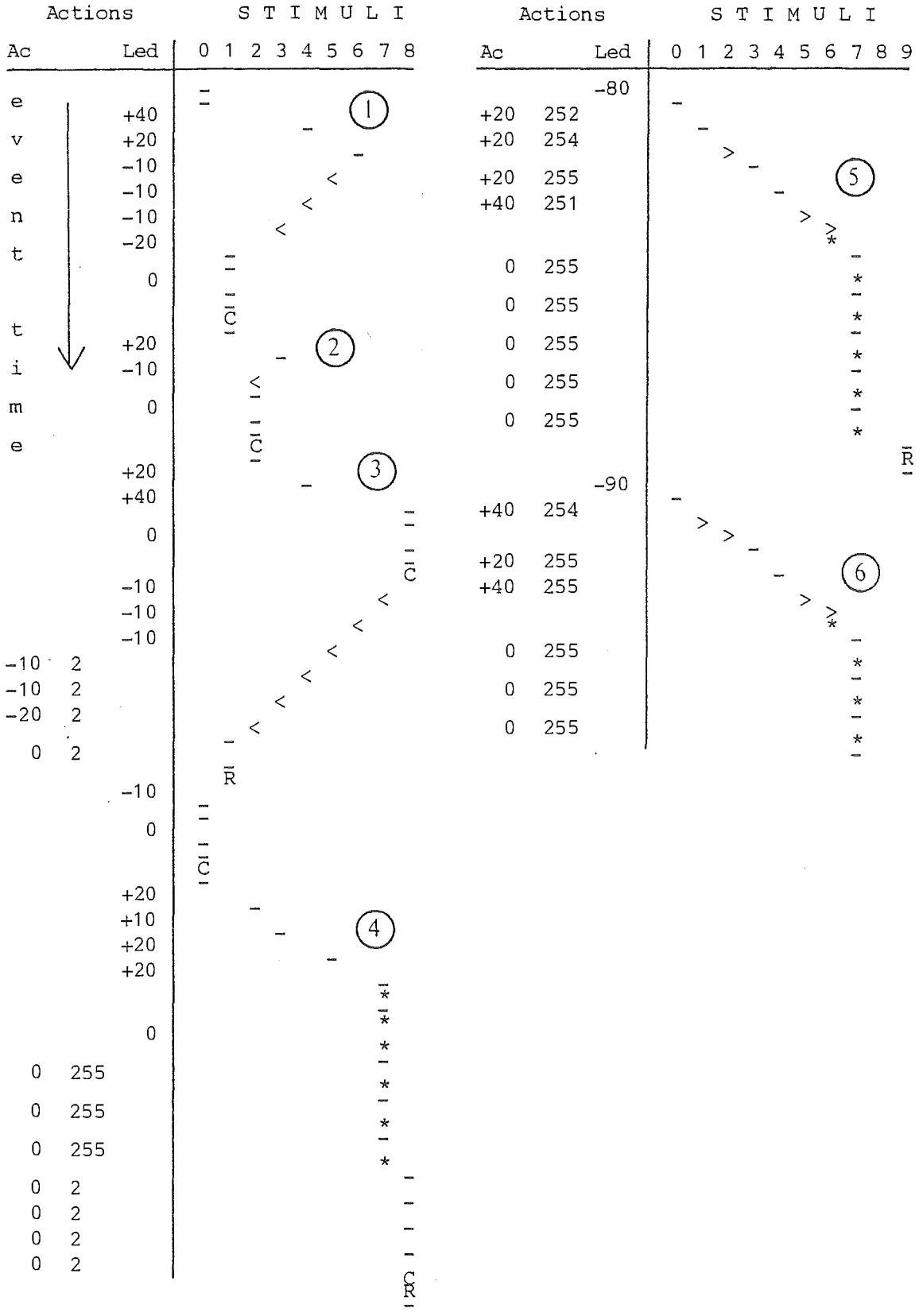
cross Path 1 and to start off into Path 3. Reward was obtained at 70 degrees. Now reward will leak back through Path 5, and Path 6, which is shown below. Path 6 will be preferred to Paths 4 and 5 because Path 6 is shorter, as shown in Figures V-9 and V-11. Figure V-10(b) shows the leak-back "tree" that results in Path 6.

Path 6	+40	+20	+40	0	0
	0 deg	30 deg	40 deg	70 deg Re	70 deg Re

In fact, with the minimal updating of leak-back pointers that there is in the arm-robot, reward will leak-back from 70 degrees S through +40 to 40 degrees S and on through +40 to 0 degrees S, although the robot could not follow this path. A better form of pointer updating would be to delete the pointer from 40 degrees S through +40 to 0 degrees S once a new pointer was stored from another context through +40 to 0 degrees S.

The robot did not have accurate information about the results of actions performed in the contexts they were led in. By performing the actions, the arm-robot obtained more accurate information, enabling better performance of the task. Section 4.2 of chapter III deals with the way inaccurate information about the results of actions can disappear. For example, information about the impossible---for the robot to do with the heavy weight on its arm---transition from 0 degrees S through the action +40 to 40 degrees S can disappear. Without this transition in the MCLS's memory, reward cannot leak-back through this "false" path and influence the decisions of the robot. In the example in Figure V-9, the effect of this "false" leak-back was not detrimental. However, an "optimal" way of (i) finding reward most quickly, or (ii) finding the most reward, cannot in general be found in the presence of such "false" information. Optimal goal-seeking is discussed in chapter VI.

Figure V-11 Interaction with simulated arm-robot Path numbers, 1 through 6, are shown in the Figure. A detailed explanation is given in the text.

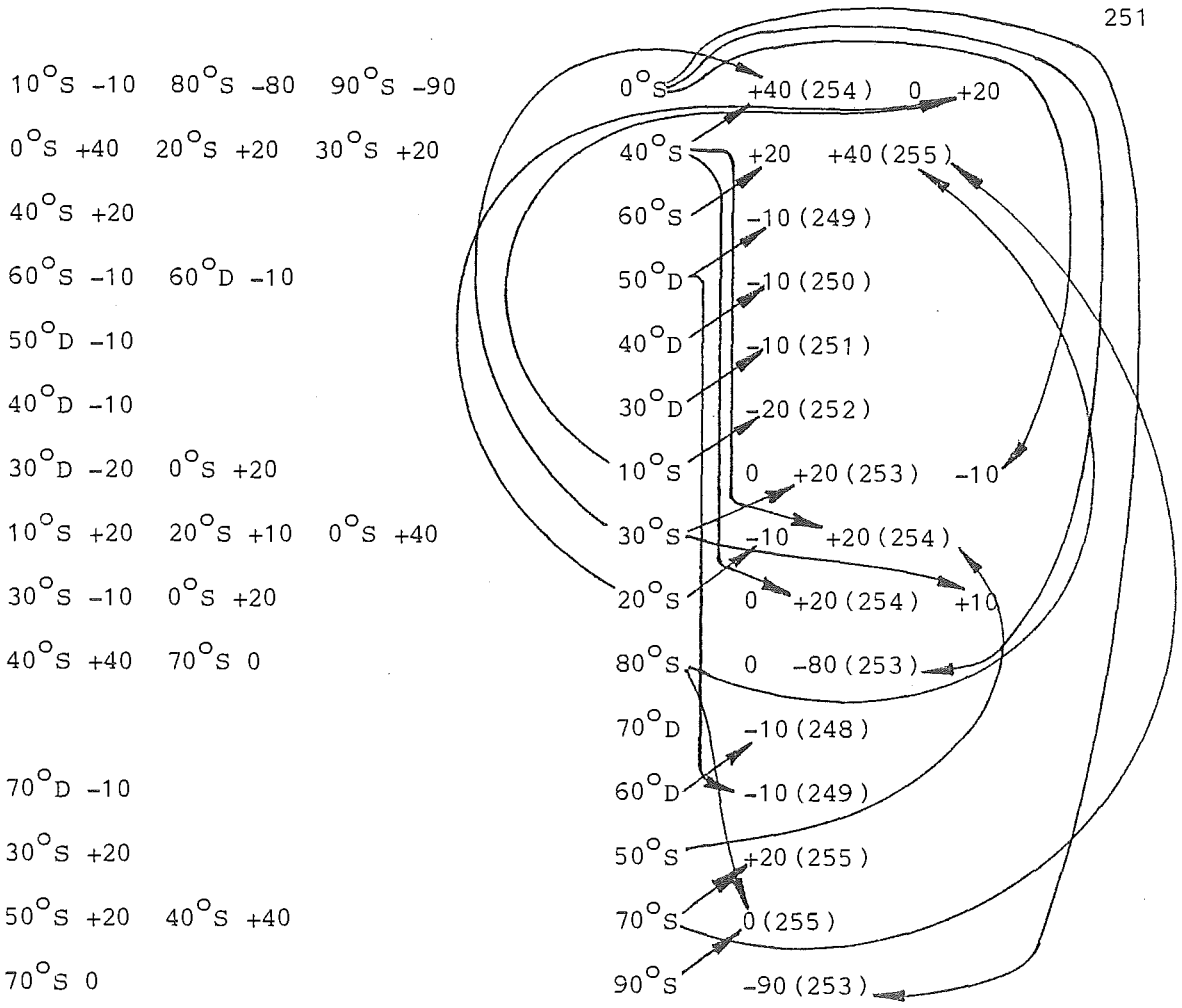


This interaction is an example of how a robot can be led through sequences of actions and stimuli---Paths 1, 2, 3 and 4---that, while they do not themselves constitute good performance of the task, enable the robot to do the task "badly"---Path 5---in a way that enables it to do the task better---Path 6---in the future.

An example of an interaction with the six paths in it is shown in Figure V-11. The example interaction has been verified with a simulated version of the arm-robot. With the exception of actions being shown in units in Figure V-11 rather than tens of units, the conventions for this Figure are the same conventions that apply to Figure V-7. The conventions are explained at the start of section A.2.1. The productions and backward links that are formed during this interaction are shown in Figure V-12.

At the end of Path 3, after the robot has learned to do a 0 or hold action at 80 degrees, the teacher lets the arm drop so that the robot learns a series of -10 actions, down to 50 degrees, where it starts doing actions itself, relaxes at 10 degrees and drops down to 0 degrees. Note that the first -10 degrees movement is led before a stimulus comes back into STM, following STM being cleared. This was intended. It prevents the action -10 being learned in the context 80 degrees S and prevents a link from 70 degrees D to 80 degrees S being formed through a -10 action. The next paragraph explains why that link is not wanted.

After the robot reaches 70 degrees on Path 4 and receives reward, it goes on doing 0 or hold actions. We want to see how the robot will reach 70 degrees from 0 degrees for a second time. So the teacher forces the arm up to 80 degrees and holds it there. The robot goes on doing 0 actions because it has learned to do a 0 action after a 0




```
+40 S  +20  0
+20 S  -10  +40  +10  +20  0
-10 D  -10  -20
-20 S  0
-10 S  0
  0 S  -10  0  -90
+10 S  +20
-80 S  +20
-90 S  +40
```

Figure V-12 Productions and Links formed during the interaction in Figure V-11

ANGLE productions are shown double spaced. ACTION productions are shown single spaced, below the ANGLE productions. ACTION productions are listed in the form "context prediction prediction ...". For example, the productions "+40 S → +20" and "+40 S → 0" are shown as "+40 S +20 0". ANGLE productions are listed in the same way on the right hand side of the Figure. On the left of each ANGLE context is a list of the productions that are linked to that context. Lines on the diagram also show these leak-back links. For example, "0° S" is linked back to the productions "10° S → -10", "80° S → -80", and "90° S → -90". Reward leaks back along these links.

Two leak-back "trees" are shown in Figure V-10. The first shows the reward leaking back after the -80 action has been led. The second shows the reward leaking back after the -90 action has been led.

action and because it has learned at the end of Path 3 to do a 0 action upon reaching 80 degrees. Both ACTION and ANGLE contexts predict a 0 action. Since reward is not being received, STM is cleared after four 80 degrees stimuli in a row. The arm then relaxes, and since there is no reward leaking back to 80 degrees S, no action is performed. ACTION context is not predicting because there is no action in STM. Teacher leads the robot to 0 degrees again. At the end of the last paragraph I mentioned that I specifically prevented the action -10 being predicted by 80 degrees S and that I prevented a link being formed to 70 degrees D. This ensures that no actions are performed after STM is cleared at 80 degrees. The arm relaxes and the teacher is able to lead it to 0 degrees.

Now there is leak-back of reward through the robot's memory from 70 degrees S to 0 degrees S as shown in Figure V-10(a). So a +20 action is performed since that is the action through which the largest value of reward leaks to 0 degrees S. +20 brings the simulated arm to only 10 degrees. A new link is formed from 10 degrees S through +20 to 0 degrees. The favoured action at 10 degrees is +20, which brings the robot to 30 degrees S. Another +20 action is performed, bringing the arm to 40 degrees S and forming a link from 40 degrees S through +20 to 30 degrees S. The action +40 is the action favoured at 40 degrees S. Reward has leaked back by a rather roundabout path through 80 degrees S and 0 S. This +40 action brings the arm to 70 degrees S and reward. The teacher pushes the arm up to 90 degrees. This causes it to relax because the ANGLE context 90 degrees S has not been seen before by the robot. The teacher leads the arm through a -90 degrees movement to reach 0 degrees S again. The leak-back paths from 70 degrees S to 0 degrees S are shown in Figure V-10(b). There have been a few added

as a result of the robot doing actions itself. Reward now leaks back from 70 degrees S directly through 40 degrees S to 0 degrees. So the robot performs a +40 action. This action brings the arm to 30 degrees S. A +20 action is performed bringing the arm to 40 degrees S. A +40 action is performed bringing the arm to 70 degrees S and reward.

By (i) the robot being led through several sequences of movements and (ii) the robot performing some of these movements itself and reaching its goal once, the robot was able to reach its goal more quickly a second time. This shows that leading a robot through movement paths, some of which it cannot follow, enables it to learn to do sequences of actions that it couldn't have been led through AND enables it to improve its performance of the task as it "tries" to do the task itself.

Figure V-13 shows two more interactions with the real arm-robot with its real metal lever arm. The conventions for Figure V-13 are the same as those for Figure V-7, which were explained in section A.2.1.

All four real interactions differ from the simulated one in three ways:

-the characteristics of the arm and load are not quite the same as those of the simulated arm and load. For example at the end of Interaction 2, when the arm-robot lifts its arm into the light beam, the arm shows this characteristic,

total input to arm	0	1	2	4	5	6	9
actual arm angle	0	0	1	3	3	4	7
simulated arm angle	0	1	1	3	3	4	6

The difference between the simulated and actual characteristics is due to stiction and hysteresis in the gears of the arm drive and variability in the torque produced by the motor when stationary.

-the real arm-robot does not perform a +40 action when the arm is at

40 degrees, while the simulated arm-robot does. This is not important for the demonstration of the arm-robot improving its performance of the task as a result of performing the task by itself. The +40 action predicted by the context 40 degrees S is not performed because it does not have more reward leaked back to it than other actions that are predicted by 40 degrees S. This in turn is due to the difficulty of producing the regular dropping movements that were obtained with the simulated interaction. In the simulated interaction reward leaks back to 40 degrees S through +40 from 80 degrees S. This path, which is discussed above, was very difficult to produce in the real interactions because it was difficult to lead the -80 action from 80 degrees to 0 degrees. Also, in Interaction 1, two +20 actions are learned instead of a +40 action, on Path 3.

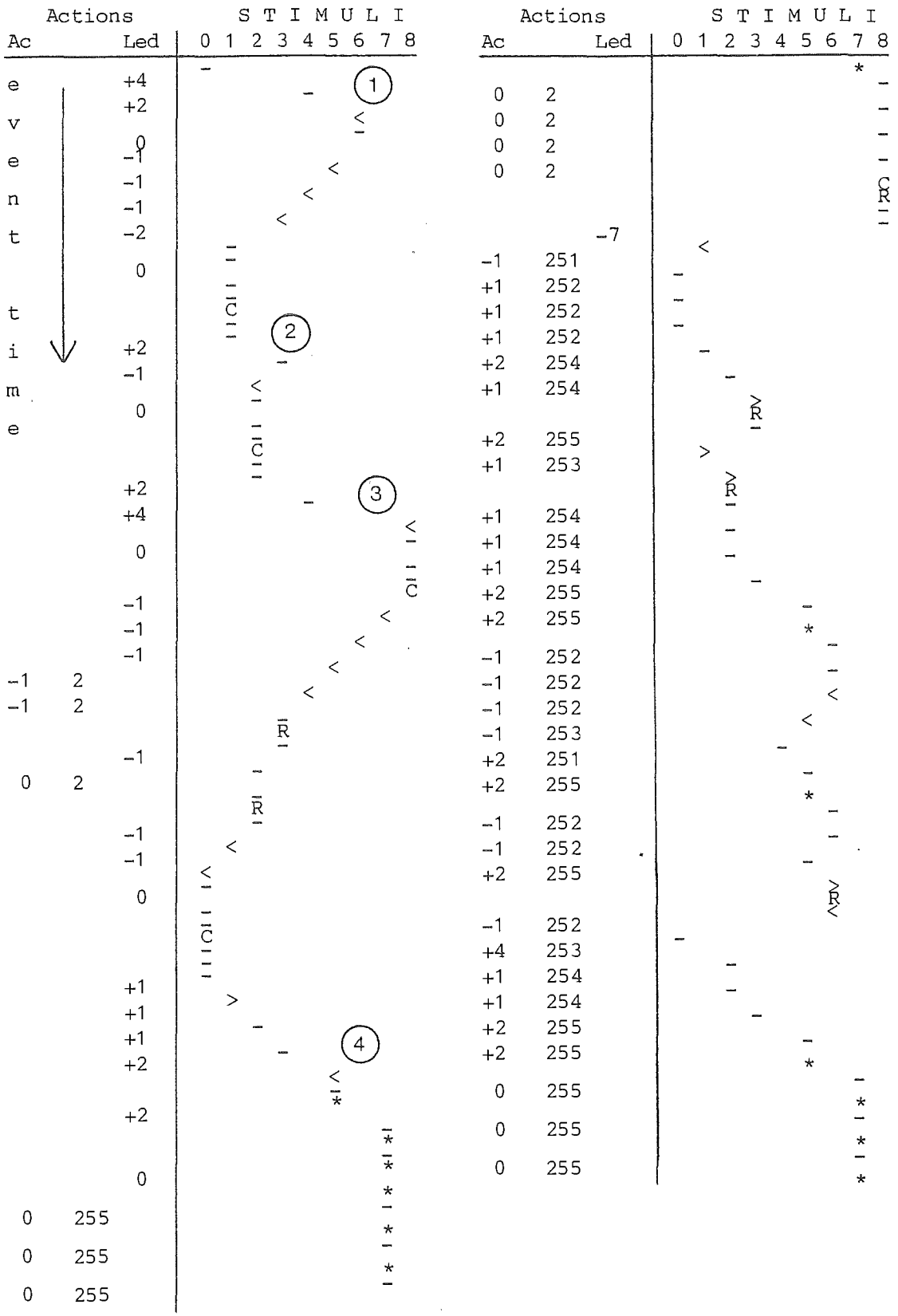
-the arm drops down very irregularly in the real interactions. This causes a variety of different sequences of actions to be learned and performed while the arm drops. For example during the first drop in Interaction 4 the arm drops from 60 degrees S to 50 degrees D, causing a -10 action to be learned, and then to 10 degrees D causing a -40 action to be learned. The teacher is leading the arm through this drop. He holds it at 10 degrees S and then leads the arm through Path 2.

INTERACTION 3

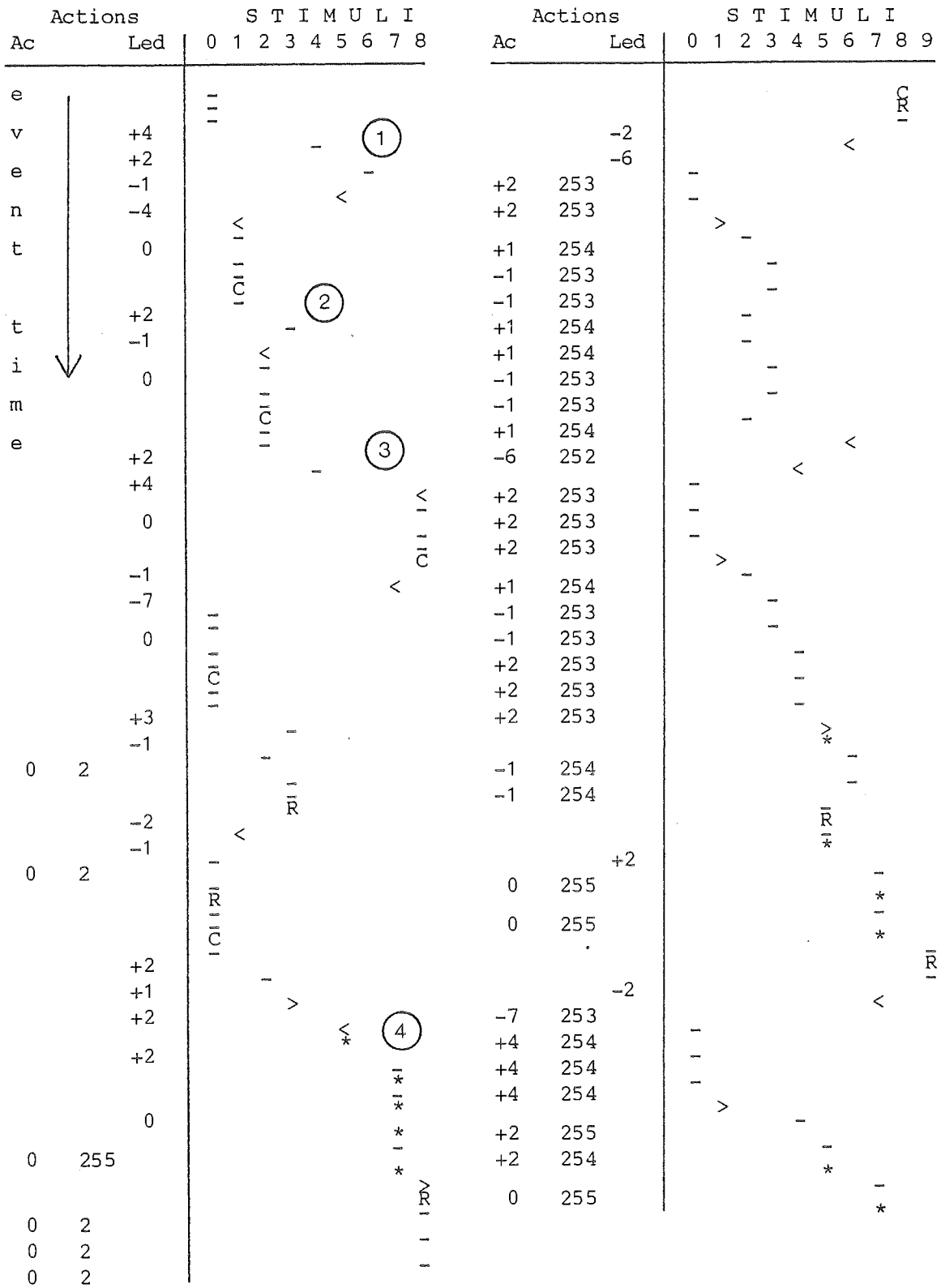
Paths 1, 2 and 3 are led as shown in the simulated interaction above. For example, the teacher starts by jerking the arm up from 0 degrees, which is shown by "-" in the "0" column of Figure V-13, to 40 degrees, which is shown by "-" in the "4" column for 40 degrees. This causes a +40 action to be learned by the arm-robot, as shown by

Figure V-13 Two more interactions with the arm-robot

(a) INTERACTION 3



(b) INTERACTION 4



"+4" in the "Led" column in Figure V-13. The +40 action is learned in the ANGLE context 0 degrees S. The teacher then jerks the arm up to 60 degrees, but he overshoots 60 degrees slightly and brings the arm back down. By the time the arm-robot registers the change in arm angle from the previous value of 40 degrees, the arm is returning from its overshoot past 60 degrees. Thus an arm velocity stimulus "D" is received by the MCLS, as shown by "<" at 60 degrees in Figure V-13. This movement from 40 degrees to 60 degrees causes a +20 action to be learned in the ANGLE context 40 degrees S and in the ACTION context +40 S.

The teacher holds the arm at 60 degrees. After three seconds another stimulus pair, 60 degrees S, is obtained. Five seconds after the +20 action was stored, a 0 or hold action is learned. The teacher smoothly guides the dropping arm down to 10 degrees in a series of three -10 actions followed by a -20 action. Now he holds the arm stationary until STM clears. The teacher goes on to lead the robot through Paths 2 and 3.

At the beginning of Path 4, two +10 actions are learned instead of the desired +20 action. The teacher jerked the arm up to 20 degrees. The arm-robot learned a +10 action to 10 degrees U and then a +10 action to 20 degrees S.

Path 4 brings the arm up to the light beam reward angle of 70 degrees, as shown by the first "*" in Figure V-13. The teacher led the arm into the light beam, obtaining reward and causing a +20 action to be learned. The teacher holds the arm in the light beam at 70 degrees, teaching the arm-robot to do a 0 action at 70 degrees. He then lifts the arm up to 80 degrees. STM clears, the arm relaxes and the teacher jerks the arm back down to 0 degrees. The robot does a -10

action itself on the way down to 0 degrees. By the time the -10 action is performed, causing the arm to unrelax, the arm has reached 0 degrees. So the -10 action causes the arm to try to go to -10 degrees. It is held at 0 degrees by a mechanical stop. Reward leaking back along Path 4 in the LTM of the MCLS causes the MCLS to do a +10 action from 0 degrees. The first +10 action "cancels" the preceding -10 action. The second +10 action does not lift the arm up to 10 degrees because of both the weight of the load and stiction in the arm drive. The third +10 lifts the arm up to 10 degrees. A +20 action is performed, as learned on Path 2. This +20 action brings the arm to only 20 degrees because of the weight of the load. A +10 action is performed, as learned on Path 4. The velocity stimulus "U" is received at 30 degrees because the arm was still moving upward when the stimulus pair was obtained by the MCLS. Before the stimulus pair 30 degrees S comes in three seconds later, the arm relaxes and starts to drop. The stimulus pair is obtained just as the arm relaxes, causing a +20 action to be performed, as learned on Path 4. By the time the +20 action is performed, the arm has dropped. The +20 action and the next action, +10, bring the arm to 20 degrees. The arm relaxes. The teacher holds the arm at 20 degrees. A stimulus pair 20 degrees S is obtained by the MCLS. Two +10 actions are performed at 20 degrees, taking the weight of the load from the teacher and enabling the third +10 action to lift the arm to 30 degrees. A +20 action is performed, as learned on Path 4, bringing the arm to 50 degrees. Another +20 action from Path 4 is performed but, because of the weight, brings the arm to only 60 degrees. Here the arm is just at the top of the 60 degree angle range. It is also just within the light beam, because the teacher mistakenly put the light beam a bit too low. Reward is obtained. Four

-10 actions are performed. The arm lifts up to 60 degrees again, and drops again. Then the arm does a +20 action and relaxes at 60 degrees. The arm drops to 0 degrees. A -10 action is performed at 0 degrees. The -10 action was predicted at 60 degrees by 60 degrees D, but the arm had reached 0 degrees by the time it was performed.

Reward has leaked back to 0 degrees through the action +40 as a result of the arm-robot "trying" to lift its arm up into the light beam. The second time the arm reached 60 degrees and obtained reward, it had come from 40 degrees. The reward value leaking back through +40 to 0 degrees is greater than that leaking back through +20 to 0 degrees because there are fewer actions along the path through +40 to reward. So the +40 action is performed at 0 degrees. This brings the arm to only 20 degrees, because of the weight. A +10 action is performed, as on Path 4. Because of the weight the arm stays at 20 degrees, so another +10 action is performed. The arm then goes on up to 70 degrees, doing two +20 action from Path 4, and obtains reward. The arm is held at 70 degrees by the arm-robot doing a series of 0 actions.

INTERACTION 4

Paths 1, 2 and 3 are led as shown in the simulated interaction. Then, afer a +30 action is led in error, Path 4 is led. On Path 4 the velocity U is received at 30 degrees, instead of the velocity S. When the arm-robot lifts its arm to 30 degrees and obtains a velocity S it does a -10 action, taught on Path 2, instead of the +20 action on Path 4. This causes the arm-robot to go into a loop, oscillating between 20 degrees and 30 degrees. The teacher breaks the loop by shoving the arm up to 60 degrees. When the arm rises to 30 degrees again the teacher lifts it up to 40 degrees, preventing the arm from dropping to

20 degrees. The arm reaches 70 degrees, after relaxing and being led through a +20 action. Reward leaks back through +40 to 0 degrees so a +40 action is performed at 0 degrees. In fact several +40 actions are performed, "cancelling" the -70 action performed before them. The -70 action caused the effective arm input command angle to be less than 0. There is a mechanical stop which prevents the actual arm angle from being less than 0 degrees.

These two interactions show the arm-robot improving its performance of the light beam task by "trying" to perform the light beam task.

V.A.3. Interactions with the earlier arm-robot

As explained in section A.5, the leak-back process in the arm-robot can be stopped. The arm-robot then becomes the earlier version; one without goal-seeking. The environment of the earlier arm-robot's interactions is different from that of the later arm-robot's interactions. The arm of the arm-robot can go down to 0 degrees, and it is rewarded at 70 degrees, while the arm of the earlier arm-robot can go to only 20 degrees, and it is rewarded at 80 degrees.

Nine interactions with the earlier arm-robot are shown in Figure V-14. In each of these interactions the earlier arm-robot acquires the light beam task and performs it both with and without a weight on its arm. The first time reward is received, the arm has no extra weight on it. Then a weight is put on the arm. The arm sags, relaxes, drops, and moves to the reward angle with the weight on it. Each of the interactions are different.

An interaction with a simulated version of the earlier arm-robot is reported in MacDonald (1981). The nine real interactions are similar to, but not the same as the simulated one. Detailed explanations of

these earlier arm-robot interactions are given in MacDonald (1981).

The following conventions apply to Figure V-14:

-the interactions are shown as a plot of stimuli versus "event time". The horizontal scale is not real time but increases by one unit each time an action or stimulus occurs, the arm relaxes, STM is cleared or reward is obtained

-a "'" means "stopped" at the angle shown on the vertical axis where the "'" is

-a "↑" means "moving upwards" at the angle on the vertical axis

-a "!" means "moving downwards" at the angle on the vertical axis

-an "R" means that the arm is relaxed at the angle on the vertical axis

-a "C" means that STM was cleared

-an "*" means that the arm-robot was rewarded after the action before the "*"

-actions are shown as single digit numbers for tens of action units. There are separate rows for negative and positive actions and separate pairs of rows for "Led" actions and "Ac" actions.

Note that the arm is more oscillatory with the weight on it; for example in Interactions 5, 6 and 7. When the arm-robot does a +20 action from 20 degrees with the weight in its arm, it sometimes gets a 40 degrees, D stimulus pair after the +20 action. The arm has overshot 40 degrees and the velocity is read as the arm drops back down to 40 degrees.

Figure V-14 Nine interactions with the real, earlier arm-robot (a)

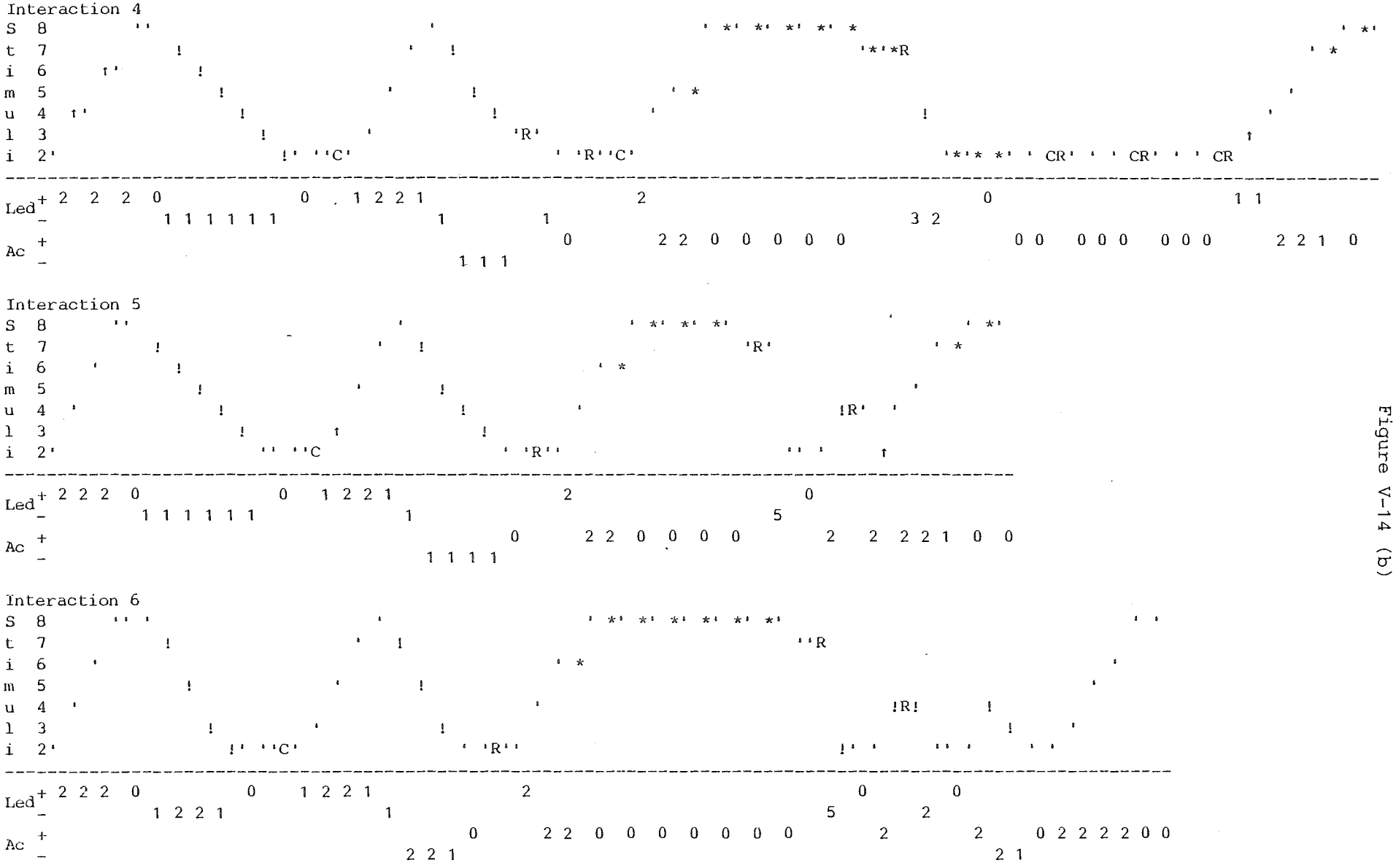


Figure V-14 (b)

Figure V-14 (c)

V.A.4 Discussion of Arm-robot interactions

This section discusses five aspects of the arm-robot and of the interactions explained in this chapter. These five aspects are discussed in sections A.4.1 through A.4.5 below. Briefly the five are:

- the extent to which the arm-robot, which has a single degree of freedom arm, meets the requirements for determining motor commands or actions for tasks (A.4.1).
- how the design of the arm-robot can be improved (A.4.2).
- how leading the robot through movements gives it a repertoire of actions for responding to information obtained about the environment (A.4.3).
- how leading the arm-robot can show the robot information about the environment as well as what movements to do (A.4.4).
- the asynchronous nature of the MCLS in the arm-robot (A.4.5).

V.A.4.1 Determination of motor commands

The requirements for determining motor commands were given in section 2 of chapter II. The state-of-the-art in achieving those requirements was also examined there. The extent to which the arm-robot satisfies the requirements is given in seven sections, (a) through (g) below:

- obtaining the required arm tip positions given the initial description of the task (a)
- obtaining the required joint trajectories given the tip positions (b)
- solving the dynamic equations of the arm in order to obtain the required actuator signals from given joint trajectories (c)
- controlling the forces and torques exerted by the arm (d)
- using sensory information (e)

- coping with a range of situations (f)
- coping with uncertainty about the environment (g)

(a) Initial description gives tip positions. It is not possible to give the arm-robot a rough English description of the light beam task, because the MCLS in the arm-robot is a very simple one which does not handle natural language. The task is specified (i) by the teacher leading the robot through movements, and (ii) by the light beam reward system.

(b) Tip positions give joint positions. The arm-robot has a single degree of freedom arm so it has only one joint trajectory and can exert a force only in the direction perpendicular to both the arm link and joint axis. So the position of the tip of the arm can be controlled in only one dimension, rather than three dimensions. The orientation cannot be controlled. The specification of the position of the tip is given by the teacher leading the arm through movements. The joint trajectory is very simply related to the tip position since there is only one degree of freedom. That is, the kinematic equations of a one degree of freedom arm are very simple. The distance the arm tip moves, along the arc in which it is constrained, is the product of the length of the arm link and the joint angle.

(c) Solving the dynamic equations. The dynamic properties of the arm are not controlled by the arm-robot. Position and velocity feedback closed around the one joint are used. There are no joint interaction forces since there is only one joint. Loads have two effects on the arm. Gravity causes the arm to sag if there is a load on the arm. Loads on the arm increase the rotational inertia of it, so

the arm takes longer to speed up and slow down. The real interactions with the arm-robot show that the arm takes longer to speed up and to slow down when it has a weight on it. These real interactions were reported in sections A.2 and A.3. The light beam task does not require the arm-robot to control the dynamics of its arm. The light beam is stationary.

(d) Controlling forces and torques. The robot can exert a force only in the direction perpendicular to both the joint axis and the arm link. It can't exert any torques. The arm-robot copes with the static effect of a load as explained in sections A.2 and A.3.

(e) Using sensory information. A primitive form of visual information is used for rewarding the arm-robot. However this does not amount to using visual information for coping with different conditions. Take for example the task of hitting a tennis ball. Information about the ball's movements is required, rather than a reward signal for actually hitting it. The arm-robot uses sensory information about the angle and velocity of its arm. This enables it to "know" when its arm has sagged. For example, if the action +40 moves the arm from 0 degrees to 20 degrees, then the arm has sagged.

(f) Coping with a range of situations. The earlier arm-robot's interactions in Figure V-14 show the robot lifting its arm into the light beam in two different situations; with and without a weight on it.

(g) Coping with uncertainty about the environment. Uncertainty in the arm-robot's environment was specifically avoided. However, as explained in sections A.2 and A.3, a few "unexpected" things happened.

For example, in Interaction 1 a 0 action is unexpectedly performed at 30 degrees on Path 4.

V.A.4.2 Improvements to the arm-robot

The following improvements to the design of the arm-robot are discussed in sections (a), (b) and (c) below:

- improvements to the design of the single degree of freedom arm (a)
- increasing the number of degrees of freedom (b)
- improving the MCLS brain of the arm-robot (c)

(a) Improving the design of the single degree of freedom arm. Changes in loads have two effects on the arm. The effect of gravity changes. The dynamic characteristics of the arm and load change. These two effects are described below, along with a method for coping with them. The method is to alter the gain of the forward control path in the second order arm control system.

The single degree of freedom arm discussed in section A.1.1 is a second order system when in its unrelaxed state. A system with a torque motor actuator and with position and velocity feedback is a second order system (Pipes & Harvill, 1970). A second order system can be described by the characteristic equation,

$$J\ddot{\theta} + R\dot{\theta} + K\theta = 0 \text{ or } \ddot{\theta} + \frac{R\dot{\theta}}{J} + \frac{K\theta}{R} = 0 \qquad (1)$$

The response of such a system to a unit step change in input signal is,

$$\frac{-w_n}{w_d} e^{-w_n z t} \sin(w_d t + \phi) + \text{step change in signal} \qquad (2)$$

where $w_d = w_n \sqrt{1-z^2}$, $\frac{K}{J} = w_n^2$, $\frac{R}{J} = 2w_n z$ and $\phi = \tan^{-1}(w_d/w_n z)$

(Di Stefano et al, 1967). $\ddot{\theta}$ is the acceleration of the arm link, $\dot{\theta}$ the velocity and θ, the joint angle, is the angle of the arm link. J is

the moment of inertia of the arm and its load. K is the position feedback gain, as shown in Figure V-3 . R is the velocity feedback gain as shown in Figure V-3. w_n is the radian frequency the sinusoid would have if the damping factor were zero. w_n is called the undamped natural frequency of the system. w_d is the radian frequency of the damped sinusoid and is called the damped natural frequency of the system. z, often denoted by the greek letter zeta, is called the damping factor of the system. The percentage overshoot in the step response depends on z. The first term in (2) is a damped transient sinusoid. The exponential damping rate is $w_n z$. The frequency of the sinusoid is w_d.

The effects of load changes on the dynamic characteristics of the arm are discussed immediately below. The gravitational effect of load changes is also discussed.

From (1), if J increases by nine times then R/J and K/J both decrease by nine times. This change would cause w_n to decrease by three times and z to decrease by three times. If z had been 0.7 and w_n had been 1 radian per second then the changed values would be 1/3 for w_n and 0.23 for z. For these values w_n would be 0.32. That value of z gives 50% overshoot in the step response. If z is 0.7 then the overshoot is only 5%. Since $w_n z$ has been reduced by nine times, the transient term in (2) takes nine times as long to die out. For a nine-fold increase in the moment of inertia of the arm and its load there is an increase from 5% to 50% overshoot and a transient which persists for nine times as long.

The moment of inertia, J, of a thin bar about one end is approximately given by one third of the product of the mass of the bar and the square of the length of the bar (Spiegel, 1968). That is,

$$J = 1/3 \text{ (Mass} \times \text{Length}^2) \qquad (3)$$

Imagine that the arm link weighs 2 kg. A 16 kg load distributed over the length of the arm would increase the moment of inertia of the arm by nine times. The moment of inertia contribution of a load at the end of the arm is given approximately by the product of the mass of the load and the length of the arm squared. That is,

$$J_{load} = \text{Mass}_{load} \times \text{Length}^2 \qquad (4)$$

The moment of inertia of the arm and load together would be the sum of their individual moments of inertia. So a 16/3 kg, or 5 1/3 kg load placed on the end of the 2 kg arm would increase the moment of inertia of the arm by nine times.

To summarize the last two paragraphs, if the link of this arm were to weigh 2 kg then a load of 5 1/3 kg placed on the end of it would cause an increase from 5% to 50% overshoot in the step response of the arm. The time for which the transient lasts would be increased by nine times. Since the natural damped frequency, w_d, is also reduced, the time for the arm to speed up and slow down would be increased. For many tasks such a change in the dynamic response of the arm system would be unacceptable. For example the spray-painting task requires that the motion of the arm be controlled continuously in time and space (McGhee, 1979). If a robot were to do the spray-painting task with a spray gun and paint can in its hand then it would have to cope with the change in mass as the paint was used up. Of course special purpose painting arms do not have a can on the arm tip (see Haugan, 1974; Haugan & Jarvis, 1974).

In order to keep the dynamic characteristics of the arm constant, despite load changes, the ratios K/J and R/J in (1) would have to be kept constant. Thus if J increased by n times then the values of K and

R would have to be increased by n times.

In considering only (2) as the solution to (1), I have assumed that the second order arm system has a damping factor less than unity. If the damping factor were greater than unity then the solution to (1) would have no sinusoidal component. However, for the damping factor of an arm control system to be greater than unity the system must be heavily damped. A heavily damped arm system would be slow moving and therefore of no use for tasks requiring rapid well-controlled movements; for example, chopping wood. When a heavily damped system is loaded it becomes even slower, just as a less damped system slows down when loaded.

For an arm to be stable against gravity it must provide more disturbance correcting torque than the change in the gravitational torque produced by the disturbance (Benati et al, 1980). Otherwise it will be unstable like an inverted pendulum, as shown in Figure II-10. For a single degree of freedom arm that moves about a horizontal axis the torque produced by gravity is,

$$gL(M_a/2 + M_l) \sin\theta \qquad (5),$$

where g is gravitational acceleration, L is the length of the arm, M_a is the mass of the arm and M_l is the mass of the load on the end of the arm. θ is measured from vertically downward. Figure II-10 shows the gravitational torque, T_g. The torque, T_a, produced by the arm when there is a step disturbance, $\delta\theta$, in arm position is,

$$K \times \delta\theta \qquad (6)$$

because with the arm stopped the error signal is the product of the position error gain, K, and the error in position, $\delta\theta$. δ means "change in". The arm torque is also shown in Figure II-10. For the arm to be stable under the influence of gravity, the correcting torque

given by (6) must be greater than

$gL(M_a/2 + M_1)\ \delta[\sin\theta]$ (7)

so that the arm control system can always bring the arm back to its equilibrium position. Since $\sin\theta$ is approximately θ if θ is small, and since $\delta\sin\theta$ is greatest when θ is small, (7) means that for the arm to always be stable against gravity, the condition,

$K > gL(M_a/2 + M_1)$ (8)

must always be satisfied (Benati et al, 1980).

Note that if M_1 is much greater than M_a then (8) becomes,

$K > gLM_1$ (9).

Now in order to keep the dynamic characteristics of the arm constant we wish to keep K/J and R/J constant. If K and R were increased in proportion to J then (9) would be kept satisfied. If K is just greater than gLM_1 then K/J becomes about,

$gLM_1/M_1L^2 = g/L$, a constant (10)

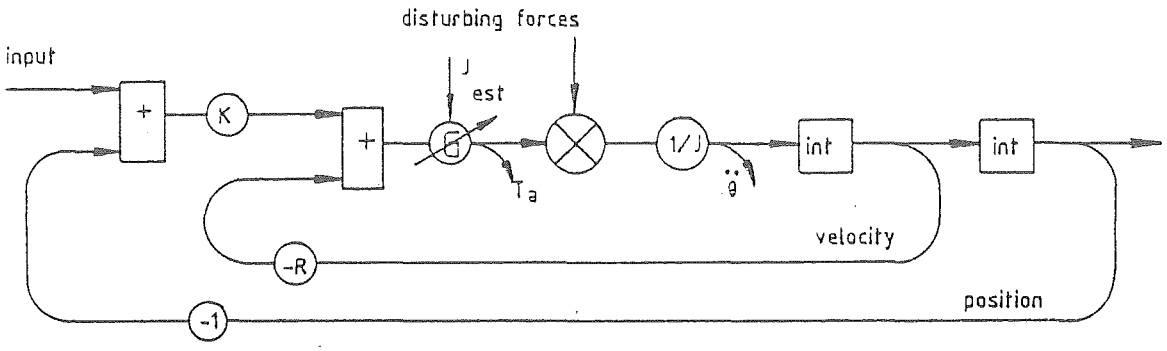
if M_1 is much greater than M_a. So if K and R can be adjusted in rough proportion to J then stability and constant dynamic characteristics can be guaranteed. The no load value of K must satisfy (8) with M_1 equal to zero. The diagram in Figure V-15 shows a method for adjusting the effective values of K and R.

It is necessary to estimate the value of J, the moment of inertia of the load and arm, in some way. If the external torques on the arm are constant then J can be obtained from (Raibert, 1978),

$\delta T_a / \delta\ddot{\theta}$ (11)

However, the torque, T_g, produced by gravity will not be constant if the arm angle, θ, is changing. But if the change in gravitational torque is much less than the change in torque produced by the joint actuator then (11) will be a good estimate of the moment of inertia of

Figure V-15 Adjusting the forward gain, G, of the arm control system in order to keep the effective values of the R and K constant. The estimate of moment of inertia, J_{est} is $\delta T_a / \delta \ddot{\theta}$. "int" stands for "integrator".

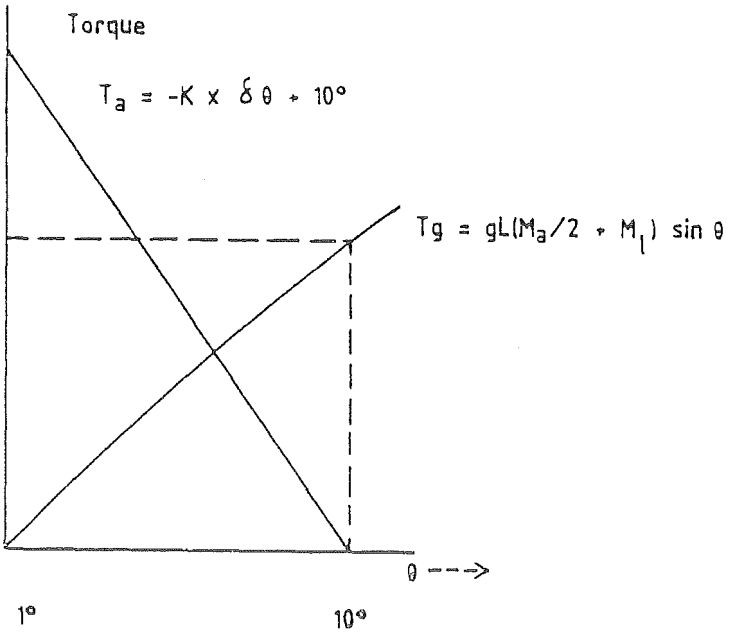


the arm and load. We already know that for stability the value of K must be greater than the maximum slope of the gravitational torque characteristic, from (8). If the input signal to the arm is a step, say equivalent to a ten degree change in arm angle, then the torque initially produced by the joint actuator will be K times the angular step, for example K times ten degrees. So we know that the change in actuator torque initially produced will be greater than the change in gravitational torque over the whole ten degrees of arm movement. This is shown in Figure V-16. If the change in acceleration of the arm were measured over the first degree of arm movement then it is guaranteed that the change in acceleration produced by gravity would not be more than ten percent of the change produced by the arm actuator. This is also shown in Figure V-16. In this way one could reasonably hope to estimate the moment of inertia of the arm and its load. Therefore it should be possible to adjust the forward gain in rough proportion to J as shown in Figure V-15. This would ensure stability against gravity and keep the dynamic characteristics relatively constant.

Note that K/J is at least g/L, from (10), if the mass of the load, M_l, is much greater than the mass of the arm, M_a. For a 40 cm arm link (10) gives a value of 25 for K/J. Hence the damped natural frequency, w_n, of the system would be 5 radians per second. Di Stefano et al (1967) report that ten percent of a step change, for example 1 degrees in 10 degrees, takes 0.4/5 seconds or 80 ms for a system with a damped natural frequency of 5 radians per second. The damping factor, z, must be between about 0.2 and 1.0 for the ten percent change to take about 80 ms. The scheme outlined above would have about 80 ms to measure the acceleration change for the 40 cm arm.

Now we also know that an arm actuator will always be able to produce

Figure V-16 Swamping the effect of gravity. If equation (8) is satisfied then an actuator torque of K x 10° will swamp the gravitational torque change over 1° of arm movement.



the torque required above for any weight that it can hold against gravity. If the actuator can produce enough torque to hold the arm and load horizontally then it can swamp gravity as well, and easily. Figure II-10 shows that the torque produced by gravity when the arm is at 90 degrees is much greater than the gravitational torque produced at 10 degrees. So we know that if the torque motor is one that can hold the load horizontally then it is also quite capable of producing the torque required to swamp the gravitational torque changes at near vertical angles. The gravitational torque is changing fastest when the arm is near vertical so this is the "worst" care for estimating the moment of inertia, J. For any load that can be held horizontally, it is guaranteed that the actuator is capable of producing enough torque to make a good estimate of J, in the way described above and shown in Figure V-16.

Nishimoto et al (1983) describe a robot controller whose velocity and position gains are adjusted so as to minimize both the integrated error and the energy expended.

(b) Increasing the number of degrees of freedom in the arm. In order to position and orient a manipulator in three dimensional space at least six degrees of freedom are required. The problems of joint interaction, kinematic equation solution, torque control and force control would all be introduced by the addition of several degrees of freedom. While it is possible to demonstrate important principles with even a single degree of freedom arm, a real robot for doing tasks like riding a bicycle, chopping wood and playing tennis would require an arm with several degrees of freedom.

(c) Improving the MCLS brain of the arm-robot. The arm-robot MCLS used in this chapter was designed for the demonstration

interactions given in sections A.2 and A.3. The design of more general MCLSs is discussed elsewhere (Andreae, 1977a; Andreae, 1972-1983). However this demonstration MCLS does have ramifications for the design of a general MCLS. Some of these ramifications are discussed in section A.4.5 below.

V.A.4.3 Leading a robot gives it a repertoire of movements

In Interaction 1 the leading of Path 4 enabled the arm-robot to respond to a particular situation that occurred while it was lifting the weight on Paths 5 and 6. The particular situation was that of only reaching 50 degrees having done a +20 action at 40 degrees. The weight caused the arm to sag to 50 degrees. Without the weight the arm would have reached 60 degrees. It would not be possible for a teacher to lead a robot through a +20 movement from 40 degrees to 50 degrees because there is only 10 degrees difference. The teacher is affecting the interaction between the robot and the weight on its arm by taking the load of the weight. In the same way a teacher could not lead a robot through the exact movements for balancing a bicycle because his leading affects the balance of the bicycle. The teacher is affecting the interaction between the robot and the bicycle. Path 4 enabled the arm-robot to respond to information about the environment and perform the light beam task. The production 50 degrees, S ---> +20 enabled the robot to do +20 after doing a +20 and only moving the ten degrees from 40 degrees to 50 degrees. In the same way a robot might be able to balance a bicycle if it were led through balancing movements. The teacher would lead the robot through steering movements. The robot might be able to respond to information about the balance of the bicycle and perform the actions required for keeping the bicycle balanced. The teacher

shows the robot a repertoire of movements, giving it a repertoire of actions.

I suggested in section A.3.2.1 of chapter IV that a teacher would find some tasks very difficult to lead a robot through because of the complex movements and subtle timing involved in the tasks. Take for example the task of hitting a tennis ball. However a teacher may be able to show the robot a repertoire of movements for doing the task, just as the arm-robot was shown movements for responding to the situation of only getting to 50 degrees having done a +20 action at 40 degrees.

For difficult tasks and tasks where the teacher significantly affects the robot-environment interaction it may be difficult or impossible for a teacher to lead the robot through a series of movements for the tasks. For these tasks it may turn out that a major part of a teacher's leading a robot through movements will be giving the robot a repertoire of movements, rather than giving it a sequence of movements for repeating.

V.A.4.4 Leading a robot can show it information about the environment

In chapter III I used leading as a way to show the robot what movements to make, and what goals to achieve. For showing movements, a teacher moves the robot's arm around while the arm is relaxed, doing the movements for it. The robot remembers the movements. Moving a robot's arm around may also show it information about the environment, which may help the robot to achieve its goal. That is, as well as giving the robot actions to do, leading the arm around may show it stimuli. A teacher might lead a robot through a task in order to show the robot what "it feels like" rather than the

movements to do. For example, imagine that a teacher has led a robot through a variety of wrist movements so that the robot has a large repertoire of wrist actions. The teacher might then slowly lead the robot through the process of hitting a tennis ball, in order to show the robot some of the stimuli that it should get when doing the movements itself. Leading the robot to show it the stimuli it should receive may help the robot to decide which actions to do for the task.

Grossman and Taylor (1978) discuss a method for showing a robot about its environment by using the arm itself for making measurements. The process of leading the robot's arm to various positions in the environment gives the robot information about objects around it.

V.A.4.5 The asynchronous characteristics of the MCLS brain of the arm-robot

The MCLS in the arm-robot does not have the synchronous nature that other MCLSs have had (see Andreae, 1977a; MacDonald & Andreae, 1981; Andreae, 1972-83). Previous MCLSs inputed stimuli and outputed actions synchronously. They have a time period during which all actions and stimuli are treated as happening simultaneously. Stimuli are paired with actions. If there are parallel streams of actions and stimuli then the streams are synchronised. MCLSs with parallel streams of events are discussed in Andreae (1981; 1980a; 1980b), and in MacDonald & Andreae (1981).

The problem with MCLSs having this synchronous characteristic is that the real world does not work in a synchronous way. Things happen at almost any time. Different sorts of event are not synchronised. For example speech actions and stimuli may not be

closely synchronised with eye movements. Andreae (1980b) has suggested that speech actions and stimuli be chunked together in the context of a production. Null actions and stimuli are treated in a way that attempts to avoid a series of synchronised null events being more than a long null or gap. Six time periods worth of nulls are not much different from seven time periods worth of nulls, for example.

The MCLS in the arm-robot in this thesis deals with actions and stimuli in a partly asynchronous fashion. Stimuli are given to the MCLS whenever either the arm angle changes or three seconds have gone by with no change in arm angle. Every time a stimulus occurs a prediction is attempted and an action performed if one is predicted strongly enough. Thus the MCLS can respond to stimuli whenever they occur, rather than having to force them into time slots in action-stimulus pairs. Actions and stimuli do not have to be paired. Previously, if an action was performed and several stimuli obtained but no actions performed between them, then null actions or "back-reflex" actions would be generated between each stimulus. Actions and stimuli are paired. Back-reflex is discussed in chapter VII and by MacDonald (1979). An early form of it called "mimic speech" was used by Andreae (1977a). The MCLS discussed here however holds the last action until another one is performed. This enables the MCLS to do a +20 action at 0 degrees, obtain a stimulus pair 10 degrees, U, and having not predicted an action with STM being 10 degrees, U, +20, eventually get the stimulus pair 20 degrees, S. STM is then 20 degrees, S, +20. Then the action +10 can be predicted and performed as shown in Interaction 4 in Figure V-13(b), after Path 4.

Actions and stimuli are not synchronised in the arm-robot MCLS. Actions and stimuli come into STM when they happen and stay until another comes, or STM is cleared. The timing of actions and stimuli is controlled mostly by the environment of the MCLS. The MCLS must keep up with the world.

The asynchronous nature of the MCLS in the arm-robot allows a wide variety of action-stimulus timing. For example, the teacher teaches Path 4 much more slowly than the arm-robot performs a sequence itself. The time for a single movement is actually longer when the arm-robot does the movement, because the teacher does the single movements very rapidly. However the series of movements is performed much more quickly by the robot than the teacher because there are shorter gaps between actions. An example of the extreme case of this slow leading occurs on Path 4 of Interaction 3. The robot gets a stimulus pair with downward velocity when the arm gets to 50 degrees. When this happens the teacher waits three seconds for a stimulus pair with "stopped" velocity, S. Then he goes on with the teaching. When the arm-robot does a series of movements it hardly ever gets the downward velocity stimulus and so it hardly ever waits.

Palfi (1976, 1977a, 1977b) discusses the problems of having a MCLS cope with real world timing. His MCLS had a real body in the real, changing world. Palfi suggests a scheme, for coping with real world timing, which retains the "time slot" nature of the MCLS. Changes in the speed of rhythms in the world are coped with by his MCLS.

Andreae (1977a) discusses an MCLS which automatically generates null actions and stimuli when none occur for six seconds. The arm-

robot learns a hold or 0 action five seconds after learning some other action but not having any further change in arm angle. This learning of hold actions is similar to Andreae's generation of null actions. The MCLS described in section A.1 can learn only one hold action in a row. It can perform more than one in a row, as shown in all reported Interactions.

Many past MCLSs have had simulated environments, enabling synchronisation of actions and stimuli. Events are not synchronous in the real world. Two MCLSs have had real bodies in real stationary environments (CAESAR in Rushby et al, 1975; and ESAW in Palfi, 1976). Their actions and stimuli were paired into time slots. An MCLS has had a real body in a real dynamic world (SPEADY in Palfi, 1977a). Actions and stimuli were paired into time slots. The arm-robot operates in the real world. It does not pair actions and stimuli in the way that previous MCLSs do. It does not force actions and stimuli into time slots. The timing of actions and stimuli is more controlled by the world than by the robot forcing actions and stimuli into time slots.

V.A.5 Circuit diagram, modular servo system and program listings

The BASIC simulation program used for verifying the arm-robot and earlier arm-robot MCLS, and the interface to the arm, is listed in Figure V-17. The interaction in Figure V-11 was obtained with this program. The program runs on a TRS-80 microcomputer.

The program in Figure V-17 is for both the arm-robot and earlier arm-robot simulations. The program starts up in the arm-robot simulation mode. To change the program to the earlier arm-robot simulation mode these keys should be pushed:

Figure V-17 Program for simulated arm-robot

```
                    Lines 5 to 900 set up the simulation.
5 CLS:PRINT@0,"ARM-ROBOT Simulation. 1st  Dec. 1981. B A MacDonald";
70 CMD"CLOCK"
90 CLEAR 15000
100 DEFINTA-Z:I=0:Q=0:K=0:J=0:N0=0      Temporary variables.
110 DIM C1(30),C2(40,2),P1(30,6),P2(40,4)
    DIMLB(40,21),RC$(1000),LP(9)
```

Variables ending in "1" apply to ACTION productions. Variables ending in "2" apply to ANGLE contexts. Productions are stored in the C and P arrays. The contexts are stored in a list in the C array. The P array contains the predictions of the contexts in the C array. The predictions are listed in the same order as the contexts. The first of each three elements in the C2 array list is the context. The second element points to the prediction, in the corresponding position in the P array, that has the greatest reward value. The third element is the reward value of that prediction. The links back to productions that preceded contexts are stored in the LB array in the same order as the contexts in the C2 array. Each pair in the LB array points to a context in the C2 array and one of that context's predictions in the P2 array. The RC$ array is used for recording interactions. The LP array is a table of screen addresses for various light beam positions.

```
113 RC=0
120 DIM WP(9),HP(9),A(33),PL(6)
```

The WP array is a table of screen positions for displaying arm weights for different arm positions. The HP array is a table for displaying the simulated teacher's hand while the teacher is leading the arm.

```
200 'DEFINE STRINGS AND ARRAYS
210 W1$=CHR$(140)+CHR$(140):W2$=CHR$(143)+CHR$(143)    characters
                                                        for the weights
220 FORI=1TO7:READK:FORJ=1TOK:READQ:D$(I)=D$(I)+CHR$(Q):NEXTJ,I
230 D$(8)=STRING$(3,CHR$(140))+STRING$(7,CHR$(176))+CHR$(26)+    the D$
    STRING$(8,CHR$(131))+STRING$(5,CHR$(140))+CHR$(132):   array stores
    D$(9)=STRING$(24,CHR$(140))                  strings that are used to
                                                        display the arm.
235 FORI=0TO5:D$(0)=D$(0)+CHR$(191)+CHR$(24)+CHR$(26):NEXT:
    D$(0)=D$(0)+CHR$(191)
240 'ARM FUNCTION
250 READK:FORI=1TOK:READQ:A(I-1)=Q:NEXTI    array A is a table of
                        actual arm angles versus arm angle input signals
260 FORI=0TO9:READQ:WP(I)=Q:NEXTI'WEIGHT
270 PZ=5*40:RZ=4*40:SZ=3*40:RF=1    PZ is the proprioception count
                                                        down time.
                                    RZ is the relax time.
                                    SZ is the stimulus count down time.
280 FORI=0TO9:READQ:HP(I)=Q:NEXTI'HAND
290 E$="":FORI=0TO6:READQ:E$=E$+CHR$(192+Q)+CHR$(26)+CHR$(29):
    NEXT:READQ:E$=E$+CHR$(192+Q)            E$ is used for erasing the
295 FORI=0TO9:READLP(I):NEXT                                arm display
300 'DATA
310 'ARM
315 DATA 25,191,24,26,131,188,24,26,175,144,24,24,26,130,189,26,24,138,
    181,24,26,171,148,26,24,191
320 DATA 27,172,144,26,8,175,148,26,8,139,180,26,8,130,173,144,26,8,
    175,144,26,8,139,180,26,8,130,141
330 DATA 30,172,144,26,8,139,164,144,26,8,130,173,144,26,8,130,173,144,
```

Figure V-17 (continued)

```
    26,8,130,141,176,26,8,130,137,180,26,8,130
332 DATA 21,140,176,26,131,140,176,26,131,140,176,26,131,140,176,
    26,131,140,176,26,131,132
342 DATA 28,140,176,144,26,8,130,131,140,180,144,26,8,130,131,140,
    164,176,26,8,131,137,140,176,26,8,131,137,140
344 DATA 27,140,164,176,144,26,8,130,131,137,140,164,176,144,26,8,
    130,131,131,140,140,176,176,26,131,131,140,132
346 DATA 24,140,164,176,176,176,26,131,131,131,137,140,140,164,176,
    176,176,26,131,131,131,137,140,140,164
350 'ARM FUNCTION
352 DATA 34,0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,4,5,6,7,8,9,0,1,1,2,3,3,4,5,
    5,6,7,8,9
360 'WEIGHT POSITION
362 DATA 897,901,904,843,847,785,724,661,598,534
366 'HAND POSITION
368 DATA 960,899,903,905,846,783,787,725,663,599
370 'ERASE STRING
372 DATA 26,26,26,26,25,22,17,10
380 DATA 960,964,969,908,850,788,727,664,601,537
500 'SET UP
520 PRINT@680,"Relaxed";:PRINT@694,"R";:PRINT@758,"S";:PRINT@822,"P";
530 PRINT@65,"ANGLE,VELOC";:PRINT@80,"ACTION,VELOC";  ACTION context
                    has the last arm action and arm velocity in it.
                    ANGLE context has the arm angle and velocity in it.
540 PRINT@128,"Stm";:PRINT@256,"Pro";:PRINT@384,"Pre";:
    PRINT@192,"Link";
550 PRINT@980,CHR$(191);:PRINT@985,W1$;:PRINT@988,W2$;
560 OF=21:PRINT@512,D$(A(OF));      OF holds the position in the A
      array that is the start of the arm function for the weight that
      is on the arm.
590 PRINT@876,"RA";:PRINT@886,"AC";:PRINT@950,"EA";:PRINT@992,"LA";:
    PRINT@998,"SA";:PRINT@866,"RP";
RA is the led action last learned. AC is the robot action last
performed. EA is the total absolute arm angle input signal that will be
caused by the action currently being performed by the robot. LA is the
last arm angle. SA is the current arm angle. RP is the absolute arm
position that an action was last led to.

600 'STARTING VAULES
605 QL=&HFFAF:QR=&HFFB0:QS=&HFFB1  addresses for the three timers
610 I=A(OF):LA=I:CI=I:RP=I:SA=I:EA=I    CI is the current absolute arm
                                        angle input signal
615 POKEQL,0:POKEQR,0:POKEQS,SZ   zero relax and proprioception
                                countdown timers. Start stimulus timer.
620 PRINT@697,PEEK(QR);:PRINT@761,PEEK(QS);:PRINT@825,PEEK(QL);
630 RW=7    Reward is given at 70°.
640 A1=-99:S1=-99:V1=-99     impossible values
A1 is the last arm action. S1 is the last arm angle stimulus. V1 is the
last arm velocity stimulus.

650 T1=-1:T2=-1:X1=-1:X2=-1                T1, T2 are the current
                                           ACTION, ANGLE contexts
660 QK=-1:QP=-1                           QK, QP point to the last
                                          rewarded context, prediction.
670 FORI=0TO40:C2(I,1)=-1:LB(I,0)=-1:NEXT:C2(0,0)=-99
800 HS=9:LS=0      the simulated arm cannot go above angle HS or below
810 FL=0                                                    angle LS
900 PRINT@98,"<SHIFT> (-) 1 to 9 Led";      control keys
910 PRINT@162,"A-J Re. X,Y,Z:-load "W1$","W2$", .";
920 PRINT@226,"Leak-back:- No '.', Yes '"CHR$(92)"'";
```

```
930 PRINT@290,"Rest:- S  @ 2, N  @ 0";
940 PRINT@354,"<sp> halt then go ("CHR$(94)"), Save";

main cycle

1000 PRINT@697,PEEK(QR);:PRINT@761,PEEK(QS);:PRINT@825,PEEK(QL);:
     GOSUB2000'CHECK TEACHER   display times and check for any keys
1010 GOSUB2500'GET ANGLE, VELOCITY, REWARD                  being pushed
1020 IFR=1THENRC$(RC)="Re":RC=RC+1:GOSUB 2800'STORE REWARD
1030 IFPEEK(QR)>N0THENRP=SA:PRINT@870,RP;:F=N0:GOTO1120     Check relax
                                                                 timer.
1040 IFRF<>1THENRC$(RC)="Relax":RC=RC+1:RF=1:PRINT@680,"Relaxed";
1045 IFSA=RPTHEN1100ELSERA=SA-RP     If arm relaxed and angle changed
                                                 then store a led action
1050 RP=SA:PRINT@870,RP;:POKEQL,PZ:F=1:PRINT@825,PEEK(QL);
1070 GOSUB3000'STORE LEAD ACTION IN LTM
1080 A1=RA:RC$(RC)="Led"+STR$(RA*10):RC=RC+1:GOSUB9000'UPDATE STM
1090 GOTO1120
1100 IFPEEK(QL)=N0ANDF=1THENF=N0:RA=N0:GOTO1070  Store a 0 action if
     arm relaxed, proprioception timer has reached zero and a led
     action has just been stored
1120 IFSA<>LATHENNS=0:GOTO1200ELSEIFPEEK(QS)>N0THEN1000     check for a
                                                            stimulus
1130 IFNS>=3THENNS=0:IFR<>1THENRC$(RC)="Stm cleared":RC=RC+1:A1=-99:
     S1=-99:V1=-99:T1=-1:T2=-1:QK=-1:GOSUB9020:POKEQS,SZ:
     PRINT@761,PEEK(QS);:GOTO1000
1140 NS=NS+1
1200 LA=SA:POKEQS,SZ:PRINT@761,SZ;
1210 S1=SA:V1=SV:RC$(RC)=STR$(S1*10)+CHR$(&H5B)+"o"+CHR$(&H5C):
     GOSUB9010:IFV1=0THENQ$="S"ELSEIFV1=1THENQ$="U"ELSEIFV1=-1THEN
     Q$="D"ELSEPRINT"ERROR";:STOP'UPDATE STM
1215 RC$(RC)=RC$(RC)+" "+Q$:RC=RC+1
1220 GOSUB5000'PREDICT
1230 IFPR=0THEN1000        no predictions
1240 POKEQR,RZ:PRINT@680,CHR$(199);:PRINT@697,RZ;
1250 IFRF=1THENCI=SA:RF=0
1252 CI=CI+AC:IFCI<0THENCI=0
1255 IFOF=10ANDCI>10THENCI=10
1256 IFOF=0ANDCI>9THENCI=9
1257 IFOF=21ANDCI>12THENCI=12
1258 EA=A(OF+CI)
1259 PRINT@953,EA;
1260 GOSUB6000'STORE ACTION
1270 A1=AC:RC$(RC)="Ac"+STR$(AC*10)+" ("+STR$(PR)+" )":RC=RC+1:
     GOSUB9000'UPDATE STM
1280 GOTO1000

end of main cycle

2000 'CHECK TEACHER
2005 IFPEEK(&H38FF)=0THENIFH=1THENH=0:RETURN'  ANY CHARACTER
2011 IF(PEEK(&H3880)AND1)=1THENH=1ELSEH=0:GOTO2040'  SHIFT KEY
2013 IFPEEK(&H3810)<2AND(PEEK(&H3820)AND3)=0THEN2040  leading an action
2015 I=PEEK(&H3810):FORK=0TO7:I=I/2:IFI>=1THENNEXT   numerical keys
2020 IF(PEEK(&H3820)AND1)=1THENK=8ELSEIF(PEEK(&H3820)AND2)=2THENK=9
2025 IF(PEEK(&H3820)AND32)=32THENK=K*-1       minus action led
2030 SA=LA+K:IFSA>HSTHENSA=HS
2032 IFSA<LSTHENSA=LS
2034 RETURN
2040 IF(PEEK(&H3808)AND1)=1THENOF=10:PRINT@988,W2$;:RETURN'  "X"
```

Figure V-17 (continued)

```
2041 IF(PEEK(&H3808)AND2)=2THENOF=21:PRINT@985,W1$;:RETURN'    "Y"
2042 IF(PEEK(&H3808)AND4)=4THENOF=0:PRINT@988,W2$;:PRINT@985,W1$;:
     RETURN'                                                    "Z"
2044 IF(PEEK(&H3804)AND8)=8THENIFSA>=2THENLS=2:A(0)=2:A(1)=2:A(10)=2:
     A(11)=2:A(21)=2:A(22)=2:RETURNELSEPRINT@1010,"Oh No !";:
     FORI=0TO50:NEXT:PRINT@1010,"       ";:RETURN'          "S"
2046 IF(PEEK(&H3802)AND64)=64THENLS=0:A(0)=0:A(1)=1:A(10)=0:A(11)=1:
     A(21)=0:A(22)=1:RETURN'                                    "N"
2050 IFPEEK(&H3801)<2AND(PEEK(&H3802)AND7)=0THEN2150  keys A-J
2055 I=PEEK(&H3801):FORK=0TO7:I=I/2:IFI>=1THENNEXT
2060 IF(PEEK(&H3802)AND1)=1THENK=8:ELSEIF(PEEK(&H3802)AND2)=2THENK=9
     ELSEIF(PEEK(&H3802)AND4)=4THENK=10
2065 RW=K-1:RETURN
2150 IF(PEEK(&H3820)AND64)=64THENPRINT@420,"No Leak Back";:FL=1:RETURN'
                                                                "."
2160 IF(PEEK(&H3840)AND16)=16THENFL=0:PRINT@420,"            ";:FL=0:
     RETURN
2190 RETURN                                              down arrow key
2500 IFH=1THENSV=0:GOTO2600   velocity, angle and reward
2550 IFPEEK(QR)>0THENIFLA=EATHENSA=LA:SV=0:GOTO2600ELSESA=LA+
     SGN(EA-LA):IFSA=EATHENSV=0:GOTO2600ELSESV=SGN(EA-LA):GOTO2600
2570 IFLA=A(OF)THENSA=LA:SV=0:ELSESA=LA-1:IFSA=A(OF)THENSV=0:
     ELSESV=-1:GOTO2600
2600 PRINT@512,E$;:PRINT@512,D$(SA);:IFOF=10THEN
     PRINT@985,"       ";:PRINT@WP(SA),W1$;ELSEIFOF=21THEN
     PRINT@988,"  ";:PRINT@WP(SA),W2$;
2602 IFLS=2THENPRINT@896,STRING$(8,CHR$(191));
2605 IFH=1THENPRINT@980," ";:PRINT@HP(SA),CHR$(191);:
     IFLA=RWANDSV=0THENR=1:PRINT@LP(RW)-1,"!";:GOTO2614ELSER=0:
     PRINT@LP(RW)-1," ";:GOTO2614ELSEPRINT@980,CHR$(191);
2610 IFRW=SAANDSV=0THENR=1:PRINT@LP(RW)-1,"!";ELSER=0:
     PRINT@LP(RW)-1," ";
2614 PRINT@LP(RW),"*";
2620 PRINT@995,LA;:PRINT@1001,SA;:RETURN
2800 'STORE REWARD-ONLY IF WAS PREDICTED BY CONTEXT 2
2810 IFQK=-1THENRETURN
2815 C2(QK,1)=QP:C2(QK,2)=255
2820 PRINT@388,C2(QK,0);P2(QK,QP) "!!!";:M=QK:J=0:PL(0)=QK:J=1:
     GOSUB4710:GOSUB11000:RETURN' LEAK BACK REWARD    Run leak-back
3000 'STORERA
3010 Q=RA:PRINT@880,RA;:GOSUB3040:RETURN
3040 PRINT@196,CHR$(216);:PRINT@260,CHR$(216);:PRINT@388,CHR$(216);:
     IFT1=-1THEN3100
3050 IFX1=-1THENC1(K1)=T1:P1(K1,0)=Q:P1(K1,1)=99:PRINT@273,K1;T1;RA;:
     K1=K1+1:GOTO3100
3060 FORI=0TO5:IFP1(X1,I)=QTHEN3100ELSEIFP1(X1,I)=99THENP1(X1,I)=Q:
     P1(X1,I+1)=99ELSENEXT:PRINT@1010,"ERROR-3060";:STOP
3070 PRINT@401,X1;T1;Q;:GOTO3100
3100 'CONTEXT 2
3110 IFT2=-1THENQK=-1:GOTO3300
3115 IFX2=-1THENIFK2<>QKANDQK<>-1THENLB(K2,0)=QK:LB(K2,1)=QP:
     LB(K2,2)=-1:PRINT@197,QK;C2(QK,0);P2(QK,QP)CHR$(93)K2;T2;
     ELSEELSE3122
3120 C2(K2,0)=T2::P2(K2,0)=Q:QP=0:QK=K2:P2(K2,1)=99:PRINT@260,K2;T2;Q;:
     K2=K2+1:GOTO3300
3122 IFQK=-1ORQK=X2THEN3130
3124 J=0:FORI=0TO8STEP2:IFLB(X2,I)=-1THENLB(X2,I)=QK:J=-1:
     LB(X2,I+1)=QP:LB(X2,I+2)=-1:PRINT@197,QK;C2(QK,0);P2(QK,QP)
     CHR$(93)X2;C2(X2,0);ELSEIFLB(X2,I)=QKANDLB(X2,I+1)=QPTHENELSENEXT:
     PRINT"ERROR";:STOP
```

```
3126 IFJ=-1ANDC2(X2,2)<>0THENPL(0)=X2:J=1:GOSUB4710      Run leak-back
3130 FORI=0TO3:IFP2(X2,I)=QTHENQP=I:QK=X2:GOTO3300ELSEIFP2(X2,I)=99THEN
     P2(X2,I)=Q:QP=I:QK=X2:P2(X2,I+1)=99:ELSENEXT:PRINT@1010,
     "ERROR-3130";:STOP
3140 PRINT@388,X2;T2;Q;:GOTO3300
3300 GOSUB11000:RETURN
4000 'CONTEXTS
4010 IFV1=-99THENT1=-1:T2=-1:RETURN
4020 T2=S1*1000+ABS(V1):IFV1<0THENT2=T2+10
4030 IFA1=-99THENT1=-1:RETURN
4040 T1=ABS(A1)*1000+ABS(V1):IFA1<0THENT1=T1+10000
4050 IFV1<0THENT1=T1+10
4090 RETURN
4510 FORI=0TO6STEP2:IFLB(M,I)=-1THENRETURN
4520 K=LB(M,I):PRINT@480,K;C2(K,0)CHR$(93)P2(K,LB(M,I+1))CHR$(93)M;
     C2(M,0);:GOSUB11000
4530 IFC2(K,2)=C2(M,2)-1THENC2(K,1)=LB(M,I+1):GOTO4700
4540 IFC2(K,2)>C2(M,2)-1THEN4700
4550 C2(K,2)=C2(M,2)-1:C2(K,1)=LB(M,I+1)
4560 PL(J)=K:J=J+1:PRINT@552,C2(K,2)"  J"J;:GOSUB11000
4700 NEXT:RETURN
4710 Z1=PEEK(QL):Z2=PEEK(QS):Z3=PEEK(QR)
4715 IFJ=0ORFL=1THENPOKEQL,Z1:POKEQS,Z2:POKEQR,Z3:RETURN
4720 J=J-1:M=PL(J):GOSUB4510:GOTO4715
5000 'PREDICT
5005 PR=0
5010 PRINT@552,"            ";:PRINT@448,CHR$(254);:GOSUB4000
5020 IFT2=-1THENX1=-1:X2=-1:RETURN
5030 'CONTEXT 1
5040 FORI=0TOK2-1:IFC2(I,0)=T2THENX2=IELSENEXT:X2=-1
5050 IFT1=-1THENX1=-1:GOTO5100
5060 FORI=0TOK1-1:IFC1(I)=T1THENX1=IELSENEXT:X1=-1
5100 J=0:IFX2=-1THEN5130
5105 IFC2(X2,1)=-1THENELSEL(J)=P2(X2,C2(X2,1)):PL(J)=C2(X2,2):J=J+1:
     PRINT@448,STR$(L(J-1));"R";
5110 FORI=0TO4:IFP2(X2,I)=99THEN5130ELSEIFC2(X2,1)<>ITHENL(J)=P2(X2,I):
     PL(J)=1:PRINT@452+I*3,STR$(L(J));:J=J+1
5120 NEXT
5130 IFX1=-1THEN5300
5140 FORI=0TO6:IFP1(X1,I)=99THEN5300ELSEPRINT@463+I*3,STR$(P1(X1,I));
5145 IFJ<>0THENFORK=0TOJ-1:IFP1(X1,I)=L(K)THENPL(K)=PL(K)+1:GOTO5150
     ELSENEXTK
5147 L(J)=P1(X1,I):PL(J)=1:J=J+1
5150 NEXTI
5300 IFJ=0THENRETURNELSEIFJ=1THENN=0:GOTO5320ELSEN=0:FORI=1TOJ-1:
     IFPL(N)<PL(I)THENN=I
5310 NEXTI
5320 IFPL(N)>=2THENPR=PL(N):AC=L(N)
5330 GOSUB11000:RETURN
6000 'STORE ACTION
6020 Q=AC:PRINT@890,AC;:GOSUB3040:RETURN
9000 PRINT@145,STR$(A1);:GOSUB11000:RETURN        Update STM
9010 PRINT@132,STR$(S1)","STR$(V1);:PRINT@149,STR$(V1);:GOSUB11000:
     RETURN
9020 PRINT@132,CHR$(211);:RETURN
```

subrountine to halt timers for recording.

```
11000 IFPEEK(&H3840)<>128THENRETURNELSEZ1=PEEK(QL):Z2=PEEK(QS):
      Z3=PEEK(QR):PRINT@1010,"Halt";:
```

Figure V-17 (continued)

```
11010 IF(PEEK(&H3840)AND64)=64THENPOKEQL,Z1:POKEQS,Z2:POKEQR,Z3:
      PRINT@1010,"    ";:RETURN
11020 IFPEEK(&H3804)=8THENGOSUB12000      "S"
11030 GOTO11010
```

Program to record interaction, links and productions on disc.

```
12000 PRINT@1010,"Record";:F$="A"+MID$(TIME$,10,2)+MID$(TIME$,13,2)+
      RIGHT$(TIME$,2)+"/DAT:1"
12010 PRINT@1010,F$;
12030 OPEN"O",1,F$:PRINT 1,RC-1
12040 FORI=0TORC-1:PRINT 1,RC$(I):NEXT
12050 PRINT 1,K2-1:FORI=0TOK2-1:PRINT 1,I:FORJ=0TO2:PRINT 1,C2(I,J):
      NEXTJ:FORJ=0TO8:PRINT 1,LB(I,J):NEXTJ:FORJ=0TO4:PRINT 1,P2(I,J):
      NEXTJ,I
12060 PRINT 1,K1-1:FORI=0TOK1-1:PRINT 1,C1(I):FORJ=0TO6:PRINT 1,
      P1(I,J):NEXTJ,I
12070 CLOSE:RETURN
```

Program to display and print interaction records.

```
14000 CLEAR2000:CMD"DIR":DEFINTA-Z:LINEINPUT"File to display or
      print?";F$:PRINTF$:OPEN"I",2,F$
14002 INPUT"P(rint OR D(isplay";Q$:IFQ$="P"THENPP=1ELSEPP=0
14005 PRINTF$:IFPP=1THENLPRINTF$
14010 INPUT 2,N:FORI=0TON:INPUT 2,A$:PRINTA$"  ";:IFPP=1THEN
      LPRINTA$" ";
14015 NEXT:PRINT:IFPP=1THENLPRINT
14020 INPUT 2,N:FORI=0TON:INPUT 2,Q:PRINTQ;:GOSUB14500:FORJ=0TO2:
      INPUT 2,Q:PRINTQ;:GOSUB14500:NEXTJ:FORJ=0TO8:INPUT 2,Q:PRINTQ;:
      GOSUB14500:NEXTJ:FORJ=0TO4:INPUT 2,Q:PRINTQ;:GOSUB14500:NEXTJ:
      PRINT:IFPP=1THENLPRINT
14025 NEXTI
14030 INPUT 2,N:FORI=0TON:INPUT 2,Q:PRINTQ;:GOSUB14500:FORJ=0TO6:
      INPUT 2,Q:PRINTQ;:GOSUB14500:NEXTJ:PRINT:IFPP=1THENLPRINT
14035 NEXTI
14040 CLOSE:INPUT"SAVE IN A FILE (Y/N) ?";A$:IFA$="N"THENGOTO14000
      ELSELINEINPUT"File ? ";A$:C$="COPY "+F$+" "+A$:CMDC$:GOTO14000
14500 IFPP=1THENLPRINTQ;
14510 RETURN
```

"J" to put the light beam at 90 degrees.

"Z" to take the heavy weight off the arm.

"." to stop leak-back.

"S" to put the simulated mechanical stop at 20 degrees. "S" works only while the arm angle is above 20 degrees.

The "I" key is used to put the simulated light beam at 80 degrees for the simulated earlier arm-robot interaction reported on Pages 41 to 47 of MacDonald (1981). The "X" key is used to put a small weight on the simulated arm in that interaction.

The "H" key puts the simulated light beam back to 70 degrees. The "Y" key puts the heavy weight back on the simulated arm. The downarrow key restarts leak-back. The "N" key shifts the simulated mechanical stop back to 0 degrees.

The assembler program, TIM/CMD, for the three timers, listed in Figure V-21 must be executed before running the program in Figure V-17.

The BASIC program for the MCLS and interface for the real arm-robot is listed in Figure V-18, with a few comments. The memory locations FF91H, FF92H and FF93H hold the relaxed/unrelaxed state of the arm, the arm angle and the arm velocity respectively. The three timers are in memory locations FFAFH, FFB0H and FFB1H. The assembler program SER/CMD for (i) the timers, (ii) controlling the relaxed/unrelaxed state of the arm, and (iii) controlling the A/D converter, is listed in Figure V-21. SER/CMD must be run before the BASIC program is run. In Figure V-18 there are some pairs of variables, one of a pair ending in "1" and the other ending in "2". "2" means that the variable is to do with ANGLE productions. "1" means that the variable is to do with ACTION productions. LTM is

Figure V-18 Real arm-robot program

Lines 10 to 900 set up the arm and the program.

```
10 CLEAR2000:GOSUB11:CLS:PRINT@512,"ARM-ROBOT  7th  January  1982.
   B A MacDonald";:GOTO15
11 POKE&HFF91,9                                          'Relax the arm
12 CMD"DIR":INPUT"Unique recording code";DD$:IFLEN(DD$)>6THEN
   DD$=LEFT$(DD$,6)
13 PRINTDD$:RETURN
100 DEFINTA-Z:I=0:Q=0:K=0:J=0:N0=0     'I,Q,K,J are temporary variables
110 DIM C1(122),C2(59,2),P1(122,9),P2(59,9),LB(59,21),LP(50)  'C and P
    arrays. C1 and C2 hold the contexts of the productions in LTM. P1
    and P2 hold the predictions. LB holds the leak-back pointers.
112 T1=-1:T2=-1:X1=-1:X2=-1     'T and X. T1 and T2 are the contexts in
    STM. X1 and X2 are the numbers of the contexts in C1 and C2 that
    match T1 and T2
113 K1=0:K2=0          Next free context and prediction in C and P arrays
115 U1=9:U2=9:Y1=U1-1:Y2=U2-1    U is the maximum number of predictions
                                                    per context.
120 H2=&H20:J2=0:J1=0       'H2 is the mask for obtaining the light beam
                              signal from the converter system.
270 PZ=5*40:RZ=4*40:SZ=3*40:RF=1      'Numbers of 25 ms for each timer.
 P timer times Led actions. S timer times stimuli. R is the relax timer
500 'SET UP
515 CLS:PRINTCHR$(23):PRINT@66,"Actn";:PRINT@130,"Stim";:FORI=0TO6:
    PRINT@960+I*10,CHR$(I+48);:NEXT    'Display for actions, stimuli
                                                          and timers
517 PRINT@512,"Relaxed";      ' Ports, timers, angle, velocity and relax
605 PA=&H44:PC=&H46:QP=&HFFAF:QR=&HFFB0:QS=&HFFB1:AA=&HFF92:VV=&HFF93:
    RR=&HFF91       PA O/P to D/A. PC I/P light beam. QP,QR,QS timers.
                    AA angle. VV veloc. RR relax
610 SA=0:GOSUB2000:SA=I:GOSUB2200:RP=SA:LA=SA:CI=SA  ' Initial angle
640 A1=-99:S1=-99:V1=-99 'STM starts clear.
          A1 action. S1 angle. V1 veloc.
650 DIMD(2000):DN=0        'Array D records interaction
660 QK=-1:QC=-1      'QC last ANGLE context. QK last ANGLE prediction
670 FORI=0TO59:C2(I,1)=-1:LB(I,0)=-1:NEXTI:C2(0,0)=-99
680 POKEQP,0:POKEQR,0:POKEQS,SZ       'Start stimulus timer
690 PRINT@768,"R";:PRINT@832,"S";:PRINT@896,"P";
700 SL!=.124:CO!=6.87        'Arm I/P signal = Angle * .124 - 6.87.
                    Coefficients obtained from measurements
710 BB$="    "
720 BR$=CHR$(199):BL$=CHR$(254)
730 BT$=CHR$(223)+CHR$(29)+CHR$(26)+CHR$(223)+CHR$(29)+CHR$(26)
    +CHR$(223)
800 I=SZ/16:J=PZ/16:K=RZ/16:QS$=STRING$(I,"S"):QP$=STRING$(J,"P"):
    QR$=STRING$(K,"R")      ' For displaying timers
900 LU$=CHR$(&H5B):LD$=CHR$(&H5C)

        Main cycle.

1000 IFPEEK(&H38FF)<>0THEN20000  key pushed?
1005 GOSUB2000:SA=I:GOSUB2200:I=PEEK(QR)/16:J=PEEK(QS)/16:K=PEEK(QP)/16
   Get angle and velocity. Display times
1010 PRINT@770+I*2,BB$;:PRINT@834+J*2,BB$;:PRINT@898+K*2,BB$;
1020 IF((INP(PC)ANDH2)=H2)ANDSV=N0THENGOSUB2800:R=1:D(DN)=888:DN=DN+1:
     PRINT@552,"YUM !!!";ELSEPRINT@552,"        ";:R=N0 'STORE REWARD
1030 IFPEEK(QR)>N0THENRP=SA:F=N0:GOTO1120  'Relax timer zero ?
1040 IFRF<>1THENPOKERR,9:RF=1:PRINT@512,"Relaxed";:D(DN)=777:DN=DN+1
1045 IFSA=RPTHEN1100ELSERA=SA-RP    'RA is Led action
```

```
1050 RP=SA:PRINT@898,QP$;:POKEQP,PZ:F=1    'Reset P timer
1070 Q=RA:GOSUB3000      'Store Led action
1080 A1=RA:GOSUB9000:PRINT@80,"r";:D(DN)=A1+10000:DN=DN+1:
     GOTO1120'UPDATE STM
1100 IFPEEK(QP)=N0ANDF=1THENF=N0:RA=N0:GOTO1070     'F is flag for a Led
          action other than 0 having just been learned
1120 IFSA<>LATHENNS=N0:GOTO1200ELSEIFPEEK(QS)>N0THEN1000'Angle changed?
1130 IFNS>=3THENNS=N0:IFR<>1THENA1=-99:S1=-99:V1=-99:T1=-1:T2=-1:
     D(DN)=-9:DN=DN+1:GOSUB9020:PRINT@834,QS$;:POKEQS,SZ:GOTO1000'Clear
   STM if no reward and 4 angle stimuli the same in a row
1140 NS=NS+1
1200 LA=SA:PRINT@834,QS$;:POKEQS,SZ:     'Reset stimulus timer
1210 IFSA<0THENPRINT"ERROR":STOPELSES1=SA:V1=SV:D(DN)=S1*10:DN=DN+1:
     D(DN)=V1-2:DN=DN+1:GOSUB9010'UPDATE STM
1220 GOSUB5000      'Predict
1230 IFPR=N0THEN1000
1240 PRINT@770,QR$;:POKEQR,RZ:PRINT@512,BR$;     'Reset relax timer
1250 IFRF=1THENGOSUB2000:CI=I      'If relaxed then read angle
1265 Q=AC:GOSUB3000'STORE ACTION
1270 A1=AC:PRINT@80," ";:D(DN)=A1+1000:DN=DN+1:D(DN)=PR-2000:DN=DN+1:
     GOSUB9000'UPDATE STM
1275 CI=CI+AC:IFCI>25THENCI=24ELSEIFCI<=-7THENCI=-6
1277 PRINT@58,CI;:OUTPA,256-(CI+CO!)/SL!:POKERR,8:RF=N0
1280 GOTO1000
         End of main cycle

2000 Q=PEEK(AA)*123-6616:J=SA*1000     ' Subroutine for obtaining angle
2040 IFABS(Q-J)<=600THENI=J/1000:RETURN     'Measurements gave equation,
2050 I=Q/1000:K=I*1000:IFQ>JTHEN2070          Angle = I/P * .123 - 6.616
2060 IFQ-K>400THENI=I+1                ' Hysteresis added.
2065 RETURN
2070 IFQ-K>600THENI=I+1
2080 RETURN
2200 Q=128-PEEK(VV):PRINT@118,Q;     '  Subroutine to get veloc.
2240 ONSV+2GOTO2250,2260,2270            Hysteresis added
2245 PRINT"ERROR-2245";:STOP
2250 IFQ>-3THENIFQ>5THENSV=1ELSESV=N0
2255 RETURN
2260 IFQ>5THENSV=1ELSEIFQ<-5THENSV=-1
2265 RETURN
2270 IFQ<3THENIFQ<-5THENSV=-1ELSESV=N0
2275 RETURN
2800 'STORE REWARD-ONLY IF WAS PREDICTED BY CONTEXT 2
2810 IFQK=-1THENRETURN
2815 C2(QK,1)=QC:C2(QK,2)=255:J=1:PL(0)=QK:GOSUB4710:RETURN
                                                    Perform leak-back.
3000 IFT1=-1THEN3100     'Store action
3050 IFX1=-1THENC1(K1)=T1:P1(K1,N0)=Q:P1(K1,1)=99:J1=K1:K1=K1+1:X1=J1:
     GOTO3100
3060 FORI=N0TOY1:IFP1(X1,I)=QTHEN3100ELSEIFP1(X1,I)=99THENP1(X1,I)=Q:
     P1(X1,I+1)=99ELSENEXT:PRINT@1010,"ERROR-3060";:STOP
3100 IFT2=-1THENQK=-1:GOTO3200
3115 IFX2=-1THENIFK2<>QKANDQK<>-1THENLB(K2,0)=QK:LB(K2,1)=QC:
     LB(K2,2)=-1ELSEELSE3122
3119 C2(K2,0)=T2:P2(K2,0)=Q:QC=N0:QK=K2:P2(K2,1)=99:PRINT@260,K2;T2;Q;:
     J2=K2:K2=K2+1:X2=J2:GOTO3200
3122 IFQK=-1ORQK=X2THEN3130
3124 J=0:FORI=0TO20STEP2:IFLB(X2,I)=-1THENLB(X2,I)=QK:J=-1:
     LB(X2,I+1)=QC:LB(X2,I+2)=-1ELSEIFLB(X2,I)=QKANDLB(X2,I+1)=QCTHEN
     ELSENEXT:PRINT"ERROR";:STOP
```

Figure V-18 (continued)

```
3126 IFJ=-1ANDC2(X2,2)<>0THENPL(0)=X2:J=1:GOSUB4710 perform leak-back.
3130 FORI=N0TOY2:IFP2(X2,I)=QORP2(X2,I)=Q+1000THENQC=I:QK=X2:
     GOTO3200ELSEIFP2(X2,I)=99THENP2(X2,I)=Q:QC=I:QK=X2:P2(X2,I+1)=99
     ELSENEXT:PRINT@1010,"ERROR-3130";:STOP
3200 RETURN
4510 FORI=0TO20STEP2:IFLB(M,I)=-1THENRETURN.
4520 K=LB(M,I):PRINT@640,K;C2(K,0)CHR$(93)P2(K,LB(M,I+1))CHR$(93)M;
     C2(M,0);
4530 IFC2(K,2)=C2(M,2)-1THENC2(K,1)=LB(M,I+1):GOTO4700
4540 IFC2(K,2)>C2(M,2)-1THEN4700
4550 C2(K,2)=C2(M,2)-1:C2(K,1)=LB(M,I+1)
4560 PL(J)=K:J=J+1:PRINT@712,C2(K,2)"  J"J;
4700 NEXT:RETURN
4710 Z1=PEEK(QP):Z2=PEEK(QS):Z3=PEEK(QR) Leak-back subroutine.
4715 IFJ=0ORFL=1THENPRINT@640,CHR$(191+62);:     Timers are stopped for
     PRINT@712,"      ";:POKEQP,Z1:POKEQS,Z2:POKEQR,Z3:RETURN  leakback
4720 J=J-1:M=PL(J):GOSUB4510:GOTO4715
5000 PR=N0          'Predict
5005 IFV1=-99THENT1=-1:T2=-1:X1=-1:X2=-1:RETURN
5006 T2=S1*1000+ABS(V1):IFV1<N0THENT2=T2+10   Form context 2
5008 FORI=N0TOJ2:IFC2(I,0)=T2THENX2=I:ELSENEXT:X2=-1 'Context 2 match?
5010 IFA1=-99THENT1=-1:X1=-1:GOTO5100
5011 T1=ABS(A1)*1000+ABS(V1):IFV1<N0THENT1=T1+10 Form context 1
5020 IFA1<N0THENT1=T1*-1
5060 FORI=N0TOJ1:IFC1(I)=T1THENX1=IELSENEXT:X1=-1  'Context 1 match?
5100 IFX2=-1THENRETURNELSEIFC2(X2,2)>=2THENPR=C2(X2,2):
     AC=P2(X2,C2(X2,1)):RETURN
5105 IFX1=-1THENRETURN
5110 FORI=N0TOU2:Q=P2(X2,I):IFQ=99THENRETURNELSEFORJ=N0TOU1:K=P1(X1,J):
     IFK=99THENELSEIFK=QTHENAC=Q:PR=2:RETURNELSENEXTJ
5120 NEXTI:RETURN
9000 PRINT@76,A1;:RETURN    'Display
9010 PRINT@140,S1"  "V1;:RETURN
9020 PRINT@74,"      ";:PRINT@138,"           ";:RETURN
         End of main program

10000 Z1=PEEK(QR):Z2=PEEK(QS):Z3=PEEK(QP):X=VAL(ND$)+1:
      ND$=RIGHT$(STR$(X),LEN(STR$(X))-1):F$="AR"+DD$+"/D"+ND$+":1":
      PRINT@576,F$;:OPEN"O",1,F$          Store and display interaction
10010 FORI=0TODN-1
10020 IFD(I)=-9THEND$="Stm cleared":GOTO11000
10030 IFD(I)=888THEND$="Re":GOTO11000
10050 IFD(I)=-3THEND$=LZ$+"D":GOTO11000
10060 IFD(I)=-2THEND$=LZ$+"S":GOTO11000
10070 IFD(I)=-1THEND$=LZ$+"U":GOTO11000
10075 IFD(I)=777THEND$="Relax":GOTO11000
10080 IFD(I)>5000THEND$="Led"+STR$(D(I)-10000):GOTO11000
10090 IFD(I)>500THEND$="Ac"+STR$(D(I)-1000)+
      " ("+STR$(D(I+1)+2000)+" )":GOTO11000
10095 IFD(I)<-1000THENGOTO11005
10100 IFD(I)<220ANDD(I)>-1THENLZ$=STR$(D(I))+LU$+"o"+LD$+" ":
      GOTO11005ELSEPRINT"ERROR";:STOP
11000 PRINT 1,D$
11005 NEXTI:PRINT 1,"END"     Store links and productions
12050 PRINT 1,K2-1:FORI=0TOK2-1:PRINT 1,I:FORJ=0TO2:PRINT 1,C2(I,J):
      NEXTJ:FORJ=0TO20:PRINT 1,LB(I,J):NEXTJ:FORJ=0TO9:PRINT 1,P2(I,J):
      NEXTJ,I
12060 PRINT 1,K1-1:FORI=0TOK1-1:PRINT 1,C1(I):FORJ=0TO9:
      PRINT 1,P1(I,J):NEXTJ,I
12070 CLOSE:POKEQR,Z1:POKEQS,Z2:POKEQP,Z3:PRINT@576,"              ";:
```

```
      GOTO1000
14000 CLEAR2000:CMD"DIR":DEFINTA-Z:LINEINPUT"File to
      display or print?";F$:PRINTF$:OPEN"I",2,F$
14002 INPUT"P(rint OR D(isplay";Q$:IFQ$="P"THENPP=1ELSEPP=0
14005 PRINTF$:IFPP=1THENLPRINTF$
14010 INPUT 2,A$:IFA$<>"END"THENPRINTA$"  ";:IFPP=1THEN
      LPRINTA$" ";ELSEELSEGOTO14015
14012 GOTO14010
14015 PRINT:IFPP=1THENLPRINT
14020 INPUT 2,N:FORI=0TON:INPUT 2,Q:PRINTQ;:GOSUB14500:FORJ=0TO2:
      INPUT 2,Q:PRINTQ;:GOSUB14500:NEXTJ:PRINT"  ";:FORJ=0TO20:
      INPUT 2,Q:PRINTQ;:GOSUB14500:NEXTJ:PRINT:FORJ=0TO9:INPUT 2,Q:
      PRINTQ;:GOSUB14500:NEXTJ:PRINT:IFPP=1THENLPRINT
14025 NEXTI
14030 INPUT 2,N:FORI=0TON:INPUT 2,Q:PRINTQ;:GOSUB14500:FORJ=0TO9:
      INPUT 2,Q:PRINTQ;:GOSUB14500:NEXTJ:PRINT:IFPP=1THENLPRINT
14035 NEXTI
14040 CLOSE:INPUT"SAVE IN A FILE (Y/N) ?";A$:IFA$="N"THENGOTO14000ELSE
      LINEINPUT"File ? ";A$:C$="COPY "+F$+" "+A$:CMDC$:GOTO14000
14500 IFPP=1THENLPRINTQ;
14510 RETURN
         Restart MCLS
20000 IF(PEEK(&H3880)AND1)=1THENGOTO10000ELSEIF(PEEK(&H3820)AND64)=64
      THENFL=1:PRINT@480,"No leak-back";:GOTO1005ELSE
      IF(PEEK(&H3840)AND16)=16THENFL=0:PRINT@480,"              ";:
      GOTO1005
20005 K1=0:K2=0:J1=0:J2=0:X1=-1:X2=-1:T1=-1:T2=-1:POKE&HFF91,9:A1=-99:
      S1=-99:V1=-99:GOSUB9020:QK=-1:QC=-1:PRINT@512,"Relaxed";:RF=1:
      F=0:SA=0:RP=0:CI=0:LA=0:DN=0:NS=0:C1(0)=9999:C2(0,0)=9999
20007 FORI=0TO59:C2(I,2)=0:C2(I,1)=-1:LB(I,0)=-1:NEXTI:C2(0,0)=-99
20010 POKEQP,0:POKEQR,0:POKEQS,SZ:PRINT@770,"                  ";:
      PRINT@898,"                    ";
20020 GOTO1005
```

stored in the C and P arrays. The contexts are listed in the C arrays. The corresponding position in the P array holds all the predictions for that context.

Figure V-19 shows the modular servo arm-robot and earlier arm-robot. The modular servo arm system was discussed in section A.1.1.2. The values of the gains K' and R' shown in the diagram of the arm servo system in Figure V-19, were 2 and 0.6 respectively, for all four real interactions with the arm-robot. The gains K' and R' were 0.5 and 0.1 for the real interactions 6, 7 and 8 with the earlier arm-robot. In interactions 1 through 5, and 9, they were 0.55 and 0.15. The weight on the arm was heavier in interactions 6, 7 and 8.

The converter system enables the microcomputer to communicate with the modular servo arm system. A diagram of the converter system is shown in Figure V-20. The converter system was discussed in section A.1.1.1.

The assembler program, SER/CMD, listed in Figure V-21, controls the relaxing switch in the arm and the three timers. One timer is for relaxing the arm. One timer is for stimuli. One timer is for led actions. The timing is explained at the start of section A.1.1, and in section A.1.2.3. The assembler program also controls the analogue-to-digital converter used for obtaining the angle and velocity of the arm. The assembler program is activated once every 25 ms, by a 25 ms "heartbeat" interrupt on the TRS-80 microcomputer. The program listed in Figure V-18 uses SER/CMD. The simulation program listed in Figure V-17 uses the three timers with the program TIM/CMD also listed in Figure V-21. The listing in Figure V-21 is a disassembly.

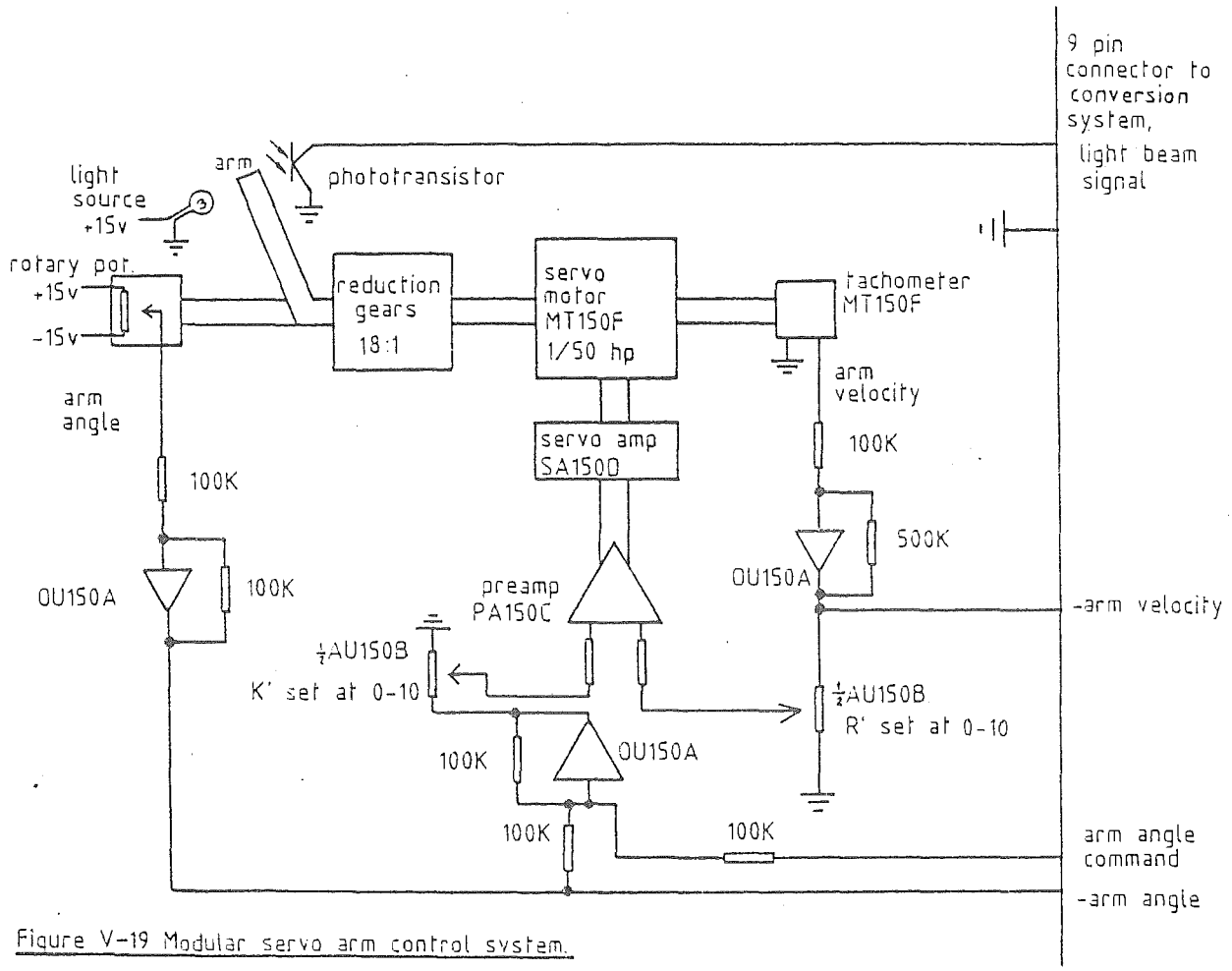


Figure V-19 Modular servo arm control system.

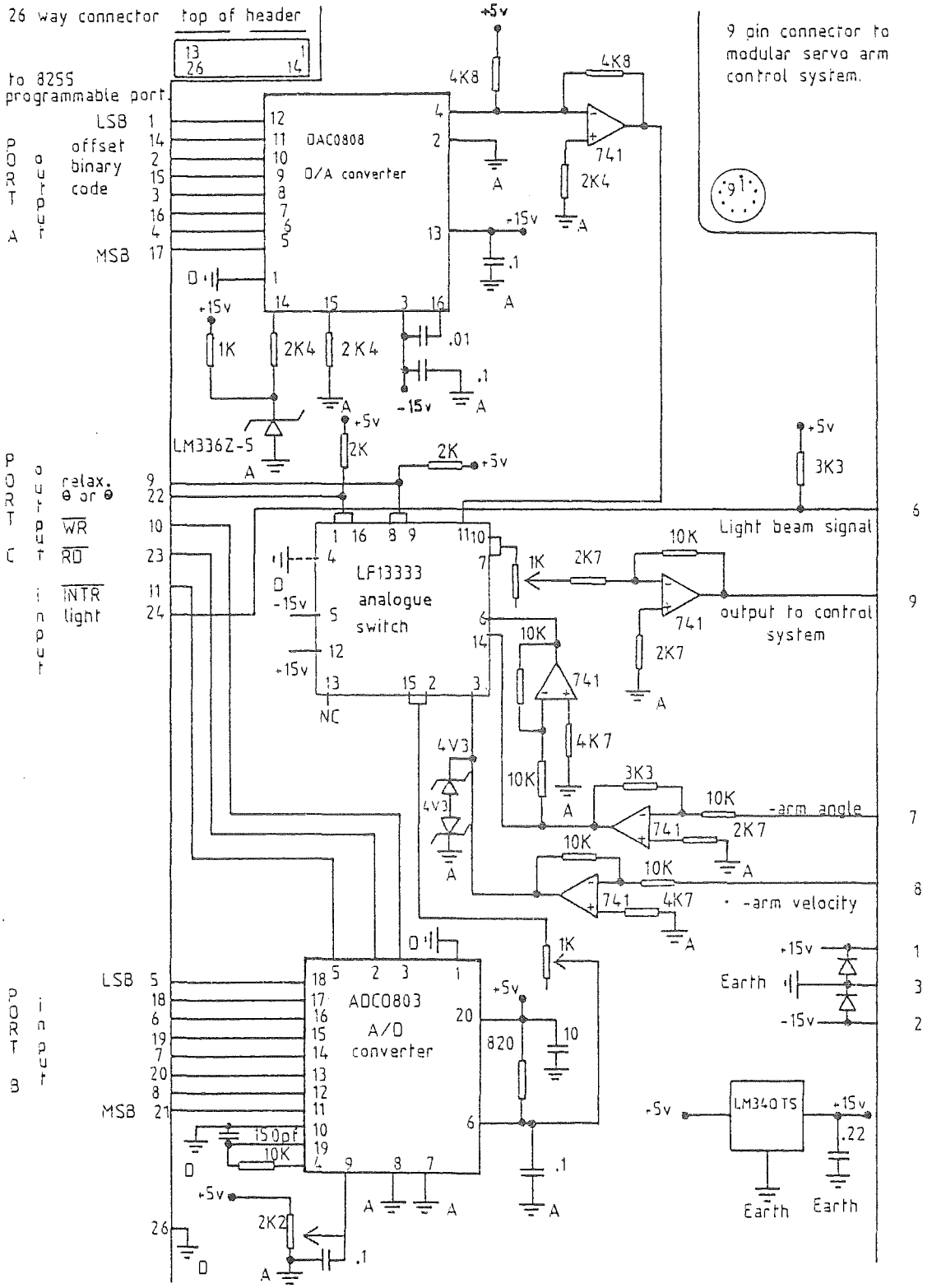


Figure V-20 Circuit Diagram of Conversion System

Figure V-21 Assembler programs

File: SER/CMD

```
  FF30 3E8A      LD   A,8AH     Set up the 8255 programmable port
  FF32 D347      OUT  (47H),A   Port  45H     input byte from A/D
                                      46H     control port for A/D and
                                             switches. Light beam signal
                                      44H     output to D/A
  FF34 3E0D      LD   A,0DH     Relax arm
  FF36 D346      OUT  (46H),A
  FF38 114DFF    LD   DE,0FF4DH
  FF3B ED534940  LD   (4049H),DE
  FF3F 13        INC  DE        Put the routine starting at FF4EH into
  FF40 CD1044    CALL 4410H     the 25 ms interrupt chain
  FF43 119DFF    LD   DE,0FF9DH  Put the routine starting at FF9DH into
  FF46 CD1044    CALL 4410H     the 25 ms interrupt chain
  FF49 C32D40    JP   402DH     Return to NEWDOS80
  FF4C 00        NOP
1 FF4D 00        NOP
  FF4E 00        NOP            Pointer used by system
  FF4F 00        NOP
  FF50 01                       number of 25 ms interrupts between calls
  FF51 01                       number of interrupts before 1st call
  FF52 0E46      LD   C,046H
  FF54 2191FF    LD   HL,0FF91H Switch arm angle to A/D converter
  FF57 5E        LD   E,(HL)
  FF58 CD65FF    CALL 0FF65H    Sample arm angle
  FF5B CBCB      SET  01H,E     Switch arm velocity to A/D converter
  FF5D CD65FF    CALL 0FF65H    Sample arm velocity
  FF60 C9        RET

2 FF65 0608      LD   B,08H     Set up a loop to take six samples
  FF67 1600      LD   D,00H     D holds the largest sample value
1 FF69 ED59      OUT  (C),E     Start a conversion
  FF6B CBD3      SET  02H,E
  FF6D ED59      OUT  (C),E
1 FF6F DB46      IN   A,(46H)   Read status of converter
  FF71 CB67      BIT  04H,A
  FF73 20FA      JR   NZ,0FF6FH Loop if not finished
  FF75 CB9B      RES  03H,E     Enable read from A/D
  FF77 ED59      OUT  (C),E
  FF79 3E05      LD   A,05H     Skip first three samples in order to
  FF7B B8        CP   B         give the A/D input time to settle. Time
  FF7C 3808      JR   C,0FF86H  constant is about 100 ms
  FF7E DB45      IN   A,(45H)   Read A/D sample
  FF80 00        NOP
  FF81 00        NOP
  FF82 BA        CP   D         Put sample in D if D is less
  FF83 3801      JR   C,0FF86H
  FF85 57        LD   D,A
2 FF86 CBDB      SET  03H,E     Disable READ
  FF88 ED59      OUT  (C),E
  FF8A CB93      RES  02H,E
  FF8C 10DB      DJNZ 0FF69H    Loop for eight samples
  FF8E 23        INC  HL        Store angle in FF92, velocity in FF93.
  FF8F 72        LD   (HL),D
  FF90 C9        RET
                                These next three locations are used by
                                    the BASIC program
1 FF91 09                       9 for relaxed. 8 for unrelaxed
```

Figure V-21 (continued)

```
  FF92 00        NOP          angle
  FF93 00        NOP          velocity

1 FF9D 00        NOP          Pointer used by system
  FF9E 00        NOP
  FF9F 01                     Number of 25 ms interrupts between calls
  FFA0 01                   Number of 25 ms interrupts before 1st call
  FFA1 21AFFF    LD   HL,0FFAFH      Timers are held in FFAFH, FFB0H
                                           and FFB1H
  FFA4 0603      LD   B,03H
  FFA6 AF        XOR  A         Clear accumulator
1 FFA7 BE        CP   (HL)      Decrement count if not already zero
  FFA8 2801      JR   Z,0FFABH
  FFAA 35        DEC  (HL)
1 FFAB 23        INC  HL        Next timer
  FFAC 10F9      DJNZ 0FFA7H
  FFAE C9        RET
  FFAF 00        NOP          Timer 1
  FFB0 00        NOP          Timer 2
  FFB1 00        NOP          Timer 3

FF30 = PROGRAM ENTRY POINT

LOCATION REFERENCE TABLE
REFERENCE SUFFIX INDICATES REFERENCING INST TYPE
  L = CALL
  P = JP
  R = JR
  S =  LD  DR,(NN)
  T =  LD   A,(NN)    ,   IN  A,(N)
  U =  LD  DR,NN
  V =  LD  SR,N       ,    OP  N
  W =  LD  (NN),DR
  X =  LD  (NN),A     ,   OUT (N),A

0000  FF67V                          0003  FFA4V
0005  FF79V                          0008  FF65V
000D  FF34V                          0045  FF7ET
0046  FF36X FF6FT                    0047  FF32X
008A  FF30V                          0E01  FF50U
2101  FF9FU                          402D  FF49P
4049  FF3BW                          4410  FF40L FF46L
FF4D  FF38U                          FF65  FF58L FF5DL
FF69  FF8CR                          FF6F  FF73R
FF86  FF7CR FF83R                    FF91  FF54U
FF9D  FF43U                          FFA7  FFACR
FFAB  FFA8R
END OF LOCATION REFERENCE TABLE
```

This program implements the timers for the simulation program.

File: TIM/CMD

```
  FF38 119CFF    LD   DE,0FF9CH
  FF3B ED534940  LD   (4049H),DE
  FF3F 13        INC  DE
  FF40 CD1044    CALL 4410H
  FF43 C32D40    JP   402DH
```

```
Lines FF9DH through FFB1H are listed above.

FF38 = PROGRAM ENTRY POINT
LOCATION REFERENCE TABLE
0003  FFA4V                          2101  FF9FU
402D  FF43P                          4049  FF3BW
4410  FF40L                          FF9C  FF38U
FFA7  FFACR                          FFAB  FFA8R
END OF LOCATION REFERENCE TABLE
```

CHAPTER VI

OPTIMAL GOAL SEEKING

In chapter III a goal-seeking (GS) system was proposed for a led robot. The network of productions in a GS system is a Markov decision process[1] (MDP. Howard, 1960). In chapter IV it was proposed that a multiple context learning system (MCLS) be used in a led robot. The network of productions in each of an MCLS's rules is also an MDP. Figure VI-1 shows the network of productions in an MDP, a rule or GS system, with the conventions that will be used in this chapter. As shown in the Figure, a Cond, in the case of a GS system, or a context, in the case of a rule, is a "state" in an MDP. The Figure shows a network of productions similar to the one in Figure III-6. However, in Figure VI-1 q(a,k(a,1)), q(a,k(a,2)), etc are different goal sizes. They are not different types of goal. Different types of goal are shown in Figure III-6. An MDP has only one type of goal. It is the active goal of the GS system or rule. When another goal becomes active, the MDP changes to a different MDP. It changes to an MDP with a different goal structure; that is, an MDP with different q(i,k(i,x))'s. In a production of a GS system in chapter III, or of a rule in chapter IV, a goal is either set, or not set. Only set goals are shown in Figure III-6. In Figure VI-1 a goal size is shown for every production. If the active goal has not been set in a production, then that production's goal size will be zero. An MDP may have several non-zero goal sizes; there may be several goals. A teacher might have set the same goal, in more than one production. So, in seeking goals a robot must make some form of trade-off between goals that may be achieved immediately, and possible future goals. Goals are sometimes called "rewards".

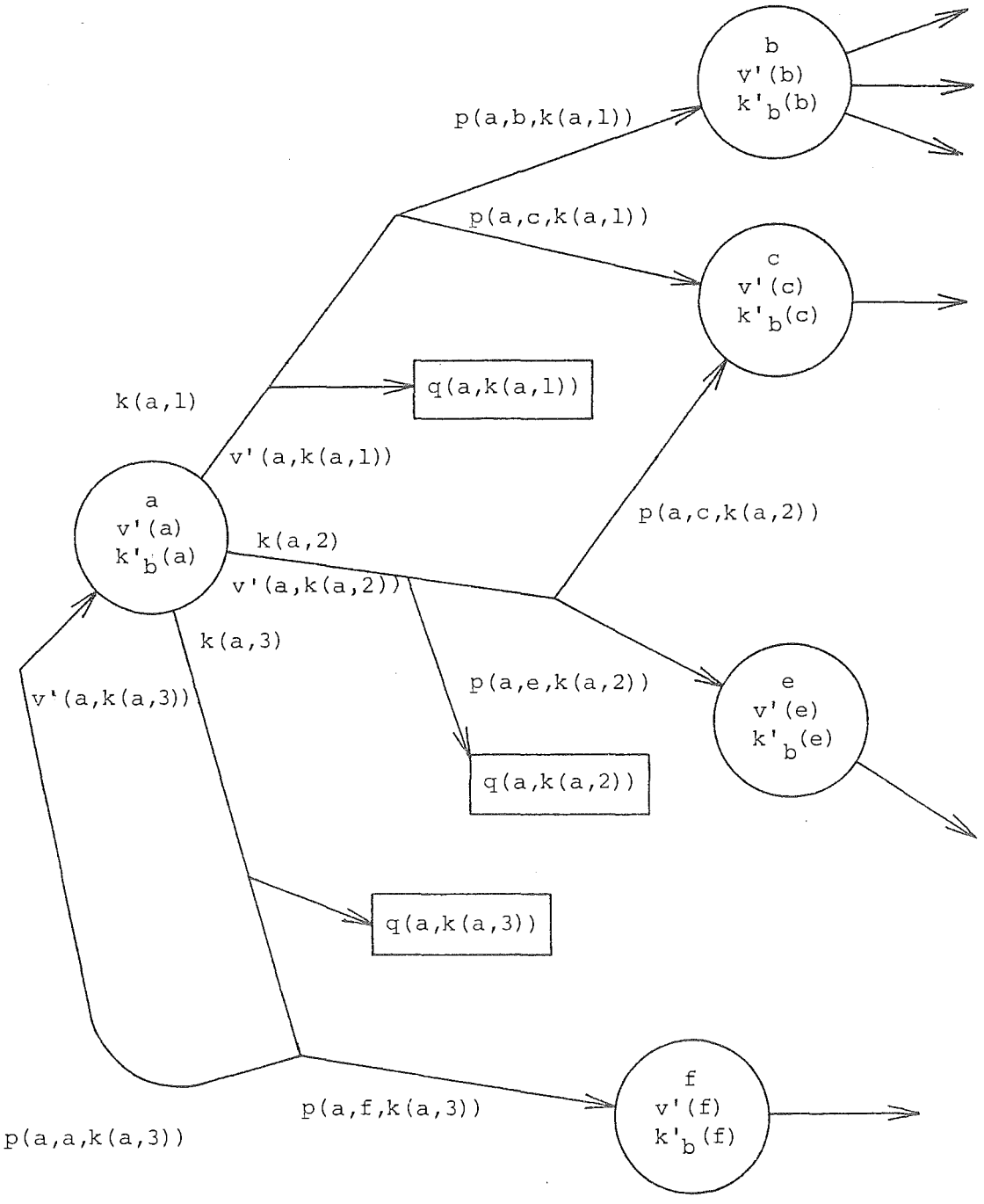
The Val of a production---in the GS systems of chapter III and in the rules of chapter IV---is v(i,k(i,x)). It is the value of performing action k(i,x), the xth available action, when in state i. v'(i,k(i,x)) is an estimate of v(i,k(i,x)) in Figure VI-1. Once the true Vals of the productions with the state i are known, the best action to perform in state i, $k_b(i)$, and the value, v(i), of the best action are known. v(i) is the value of state i, since $k_b(i)$ is the best action to perform when state i occurs. v'(i) and $k_b'(i)$ are estimates of v(i) and $k_b(i)$, respectively.

In this chapter I show that a successive approximation method, called "leak-back", can be used in a rule, that is on the system depicted in Figure VI-1, to work out $k_b'(a)$, and converge on $k_b(a)$, the "best" action, one of k(a,1), k(a,2), ... , to perform in any context a. The "best" action may be either one that optimizes the total expected future goals to be reached, or one that minimizes the number of actions to the next goal. My proof applies also to GS systems, since a GS system with an active goal is also a rule.

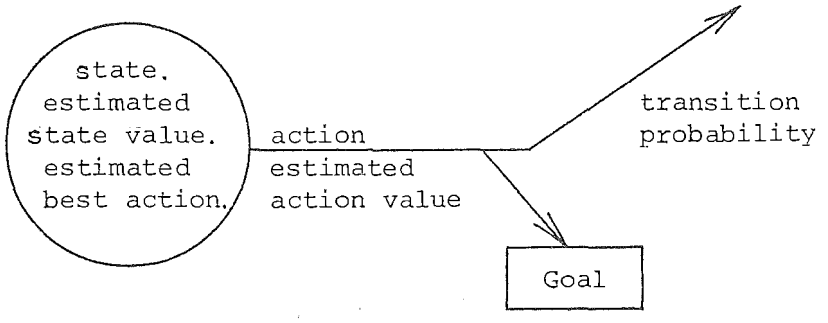
The leak-back successive approximation method of solving MDPs is equivalent to successive overrelaxation (see Van Der Wal, 1981; Reetz, 1973), with an overrelaxation factor of 1. Reetz establishes the general convergence properties of successive overrelaxation. In this chapter a specific proof of leak-back's convergence to optimal actions is given. The proof is of a different nature to that given by Reetz. My proof is along the lines of Howard's (1960) proof for a successive approximation method, called Policy-iteration, for MDPs. A policy is a set of decisions, one for each state. My proof illustrates how leak-back actually works.

Figure VI-1. A Markov decision process is shown; the internal representation in a GS system (in chapter III), and in a rule (in chapter IV). A network of productions is shown also in Figure III-6. When the system is in a particular state, or context or Cond, say a, it has a choice of actions, say k(a,1), k(a,2) and k(a,3), as shown in the Figure. With each decision is a goal size, q(a,k(a,1)), q(a,k(a,2)), and q(a,k(a,3)), which may be thought of as the immediate reward received for performing that action in that context; for example k(a,1) in context a gives immediate reward q(a,k(a,1)). Each decision has associated with it a number of transition probabilities, for example p(a,b,k(a,1)) is the probability that the action k(a,1), performed in context a, will result in the next context being context b. The value v'(a) on a context a is an estimate of the optimal total expected future reward v(a) for that context. $k'_b(a)$ is an estimate of the best action, which is actually $k_b(a)$, to perform in context a. v'(a) is the value of $k'_b(a)$. v'(a,k(a,x)) is an estimate of v(a,k(a,x)), which is the true value of performing action k(a,x), in the context a, for achieving goals. The process of "leak-back" iteratively updates the values v'(i) and best actions $k'_b(i)$ stored on each context i.

Figure VI-1 (caption opposite)



Format of Productions



Three assumptions are made about a rule in order to show that leak-back converges to an optimal policy of actions for the rule:

(a) the productions, transition probabilites and goals of the rule do not change while leak-back is operating.

(b) either (i) future rewards are discounted by a factor, d, or (ii) the optimization required is for the shortest path to the next reward.

(c) the transition probabilities for each state add up to one.

These three assumptions are explained in section 1.

Leak-back operates in parallel with the decisions of the MCLS. The MCLS does not wait for leak-back to converge. It selects the best action found so far. The MCLS can make suboptimal decisions until the optimal policy is converged on.[2] In this way leak-back at least partly avoids the "curse of dimensionality" (Bellman, 1961; Bertsekas, 1976), which afflicts the solving of MDPs. Solving an MDP at each decision step may require an impractical amount of computation.[3]

Discounting generally causes shorter paths to reward to be preferred to longer ones. However, discounting does not result in an optimally short path being followed to reward, because discounting by a factor does not penalize all transitions equally. An example of this is given in section 3.

Leak-back works within a rule. It does not enable the goals in one rule to affect predictions of another rule. A second multiple context (MC) in an MCLS can enable goals in one rule to affect the predictions of another rule. The second MC, MC-There and Then (MC-T&T. Andreae, 1980b; 1982a), or MC-Where and When (MC-W&W. Andreae, 1982b), can model the environment's responses to actions, enabling it to "go ahead" of the actual decisions of the robot. MC-T&T and MC-W&W can influence the actual predictions being made by one rule, when they meet reward in the "there and then" or "where and when", in another rule.

Leak-back was suggested by Andreae (1977a) as a chemical process in a neuron model of an MCLS. Chemical codes represent context values by their concentrations. The chemical codes leak-back through the memory of the neuron model MCLS, biassing decisions in favour of paths to goals. The leak-back method discussed here is an algorithmic version of that parallel process.

Section 1 explains the three assumptions made about an MCLS's rule, for the proof of leak-back's convergence to an optimal policy. In section 2 that proof is given. The effects of discounting future rewards are examined in section 3. Section 3 also explains why discounting is not required for finding the shortest path to a goal. The way MC-T&T and MC-W&W can cause possible future reward in one rule to affect the predictions of another rule is explained in section 4.

VI.1 THREE ASSUMPTIONS

Three assumptions are made about a rule in order to show that leak-back converges to an optimal policy of actions for the rule:

(a) the productions, transition probabilites and goals of the rule do not change while leak-back is operating. Otherwise the MDP itself would be changing in nature. That is, the optimization problem being solved would be changing while the optimization was in progress. The proof given in section 2 that leak-back finds a best policy of predictions from the productions of a rule relies on the rule being a particular, finite MDP. I stated at the start of section 4 in chapter III that the MDP that a GS system is, changes as the transition probability estimates are updated, and as productions are added. Both transition probability estimation and the addition of new productions go on in rules as well as in GS systems. So the MDP that a rule's production network is, also changes. The proof I give of leak-back's convergence in a rule applies only while the rule's production network remains the one MDP.

However, as I explain in chapter VIII, an MCLS can store and recall information from previous contexts. There an MCLS uses a large portion of its long term memory (LTM) as a working memory, a "tape". It can use the context of a production as a sort of "pidgeon hole", the prediction of the production being the contents of the pidgeon hole. The MCLS can change the prediction of the production, then retrieve the prediction later on, from the "pidgeon hole", and "look" at it. The proof given here of leak-back's convergence in a rule does not apply to rules whose productions are being used as pidgeon holes;

(b) either (i) future rewards are discounted by a factor, d, or (ii) the optimization required is for the shortest path to the next

reward. In both cases the value, v(a), of any context a will be bounded (Howard, 1971). The value of a context is the total expected future reward for the best action for that context.

If the values of contexts are unbounded then the structure of the MDP must be taken into account in order to find the best policy. There may be loops of states that, once entered, cannot be left. Such loops are called "recurrent chains" (Howard, 1971). The reward obtained in whichever of these loops is eventually entered will determine the average rate of reward in the long term, but the reward received will be unbounded. Leak-back does not take into account the structure of the MDP.

If future rewards are discounted then there is no problem in having multiple recurrent chains because the effect of rewards far into the future is negligible. In the shortest path formulation of leak-back, there are no recurrent states with reward in them (Howard, 1971). This is because only one reward is received, the one at the end of the path. The context values are bounded.

(c) the transition probabilities of each state add up to one. That is,

$$\sum_{j=1}^{N} p(i,j,k(i,x)) = 1, \text{ for all } i \text{ and all } x.$$

Andreae's (1982a) MCLSs estimate transition probabilities stochastically. The sum of a production's transition probabilities is not always one, but averages one over a long period of time.

VI.2 LEAK-BACK FINDS OPTIMAL DECISIONS

In section 2.1 I will briefly mention Howard's method of Policy-iteration (Howard, 1960; Howard, 1971). It is similar to the leak-back method. A detailed description of the leak-back method will be given. The successive overrelaxation method will also be described. In section 2.2 is given a proof that leak-back converges to an optimal policy.

VI.2.1 Policy Iteration and Leak-back

Howard's (1960) Policy-iteration method solves equation (1), below, for all of N states in an MDP. An MDP is depicted in Figure VI-1.

$$(1) \qquad v(i) = \max_{k(i,x)} \left[q(i,k(i,x)) + d \sum_{j=1}^{N} p(i,j,k(i,x))\, v(j) \right]$$

v(j) is the optimal total expected reward at state j. q(i,k(i,x)) is the immediate reward in state i given decision k(i,x). p(i,j,k(i,x)) is the probability of making the transition from state i to state j given decision k(i,x) in state i. d is the discount factor. d is greater than zero and less than one. An immediate reward unit that is one transition away is worth d times the probability of the transition. A policy, k, is a set of decisions, a set of k(i,x) values, one for each state i, for i = 1 to N. The optimal policy, k_b, is the one that maximises the total expected reward in each state. If (1) is satisfied then Bellman's (1961) Principle of Optimality is also satisfied. The Principle of Opimality says that if one chooses in state i the best decision based on optimizing the value of the very next state, then that decision will also be the best decision for optimizing the total future expected reward.

Howard's Policy-iteration method solves (1) by a two step iteration method. The "Policy Evaluation" step solves a set of linear equations,

(2), for state reward values, v'(i), given any particular policy, k. Using equation (3), the "Policy Improvement" step finds a better policy k_b' if there is one, given a particular set of reward values for each state. The iteration may begin either with an initial policy or with an initial set of values. Policy Evaluation and Policy Improvement are applied to the N states or contexts alternately.

$$(2) \qquad v'(i) = q(i,k(i)) + d \sum_{j=1}^{N} p(i,j,k(i))\ v'(j)$$

$$(3) \qquad k_b' = \max_{k(i,x)} \left[\ q(i,k(i,x)) + d \sum_{j=1}^{N} p(i,j,k(i,x))\ v'(j)\ \right]$$

Howard (1960) shows that Policy-iteration converges to an optimal policy. That is, all the v'(i) converge to v(i) and k_b' converges to k_b, the best policy.

The leak-back method also solves an MDP by iteratively solving (1). In contrast to the Policy-iteration method, Policy Evaluation and Policy Improvement are applied to the N contexts simultaneously by leak-back. Leak-back starts with an initial set of zero reward values; all the v'(i) start at zero. I will assume to begin with that reward cannot be negative. This assumption can be removed[4]. As shown in Figure VI-2, leak-back applies steps 1 and 2 below, first to context 1, then to context 2, then to context 3, ..., then to context N, then to context 1, then to context 2, ..., to context N, ..., then to context 1, ... and so on.

1. find $k_b'(i)$, the k(i,x) which solves (3) for that one context i, and update the policy k_b', with $k_b'(i)$, context i's new best action;
2. update the reward value, v'(i), of that context i according to that new $k_b'(i)$. This update is performed using (2), with the old value of v'(j) used on the right-hand side of the equation when $j \geq i$.

Figure VI-2 The two steps of the leak-back process The two steps are applied to context 1, then context 2, and so on.

STEP 1 A new best action is selected for the one context. That is, equation (3) is solved for the one context.

STEP 2 A new value for that one context is calculated, using equation (2).

The figure depicts both steps. It is important to remember that step 1 is completed first, and only then is step 2 performed.

Figure VI-2 (caption opposite)

Leak-back thus applies Policy Improvement [step 1.] and then Policy Evaluation [step 2.] to each context separately.

The way leak-back solves the linear equations (2) is a method similar to Gauss-Seidel iteration and to general numerical iteration methods (Kreyszig, 1972). In 2. the old value of context i is used in calculating the new value of context i, rather than (2) being solved directly for the value of context i. The old v'(i) is used in calculating the new v'(i). In addition, at each step of the iterative solution to the linear equations, $k'_b(i)$ for the state being "leaked back to" is updated, using (3).

The leak-back method is given by equation (4).

$$(4)\quad v'(i,n+1) = \max_{k(i,x)} \left[\begin{array}{l} q(i,k(i,x)) \\ + d \sum_{j=i}^{N} p(i,j,k(i,x))\ v'(j,n) \\ + d \sum_{j=1}^{i-1} p(i,j,k(i,x))\ v'(j,n+1) \end{array} \right]$$

first for i=1 then i=2, 3, 4, ... ,N

Equation (4) is equivalent to the method of successive overrelaxation (see Van Der Wal, 1981; Reetz, 1973). The overrelaxation factor is 1. Equation (5) expresses the method of successive overrelaxation with an overrelaxation factor of w (Reetz, 1973).

$$(5)\quad v'(i,n+1) = \max_{k(i,x)} \left[\begin{array}{l} wq(i,k(i,x)) \\ + wd \sum_{j=i}^{N} p(i,j,k(i,x))\ v'(j,n) \\ + wd \sum_{j=1}^{i-1} p(i,j,k(i,x))\ v'(j,n+1) \\ + (1 - w)\ v'(i,n) \end{array} \right]$$

Reetz (1973) shows that successive overrelaxation converges to an optimal policy for any overrelaxation factor w that is not less than zero and not greater than an optimal overrelaxation factor, w*, given by (6).

$$(6) \qquad w^* = \min_{i,x} \frac{1}{1 - dp(i,i,k(i,x))}$$

w* is always greater than or equal to 1, since both (i) d is less than 1 and greater than zero, and (ii) the probabilities are between 0 and 1. Since the overrelaxation factor in the leak-back process is exactly 1, leak-back will converge to the optimal values. It is interesting to note that the Reetz's best overrelaxation factor, w*, is 1 if p(i,i,k(i,x)) is zero for all i and x. That is, if there are no transitions that go from a state back to the same state, then the overrelaxation factor that gives the best convergence is leak-back's overrelaxation factor, 1.

The proof I give in section 2.2 of leak-back's convergence is more along the lines of Howard's proof, of Policy-iteration's convergence, than Reetz's proof. It also illustrates the leak-back process.

Note that, strictly speaking, the possibility of performing no action should be included in the possible predictions of contexts. There may be situations in which the robot will receive the most reward by doing nothing. There might be a non-zero probability that doing nothing will "result in" a transition from one context to another one. That other context might have a very high reward value. For example, suppose a robot has a goal set that is to pick up an object from a conveyor belt. The robot may need to wait for the object to come along. There must be some way for the robot to do nothing by choice. Leak-back must favour doing nothing, rather than some other action, while the robot waits for the object.

Past MCLSs have not included the possibility of doing no action in their leak-back schemes (see Andreae, 1980a; 1980b; 1982a). Some MCLSs that didn't have leak-back have been able to learn and perform a "null" action, an action that has no effect (Andreae, 1977a). The arm-robot of chapter V can learn and perform a zero or hold action, enabling it to hold its arm still.

VI.2.2 Proof: leak-back converges to an optimal policy

Leak-back begins with zero reward stored on all contexts. That is, $v'(i,1) = 0$ for $i = 1$ to N. It is assumed that all rewards, $q(i,k(i,x))$, are non-negative.

There are three parts to the proof. Let $k_b'(i)$ be the decision selected by equation (4) for context i on step n of the iterative leak-back process. I will first show that the value of that same context i under the decision $k_b'(i)$ at the (n+1)th step will either be the same as it was under $k_b'(i)$ at the nth step, or greater. I will show this for the first step ($n = 1$) and every context ($i = 1$ to N), then for the second step ($n = 2$) and the first context ($i = 1$), then the second step ($n = 2$) and the second context ($i = 2$), and so on. The values of contexts must stay the same or increase as leak-back proceeds. This first part of the proof is given in detail below.

Secondly, the estimated context values, $v'(i)$, cannot exceed the optimal values, $v(i)$. To prove this, consider a particular context's value, $v'(a)$. From (4), if all the context's values, $v'(j)$, are not greater than their optimal values, $v(j)$, then no decision in state a can put $v'(a)$ over its optimal value. All contexts start with zero values. So no context can ever have on it more than its optimal value.

For the third part of the proof, I will explain that the values given by equation (4) cannot converge to a sub-optimal policy. This third part is given below, after the first part.

It follows from the three parts that leak-back must converge to the optimal policy. The values of each context can only increase or stay the same (part 1), they cannot exceed the optimal value (part 2), and they will not stay the same if the policy is not optimal (part 3).

Now, on to the first part of the proof. The value of context i under a decision $k_b'(i)$ after the (i-1)th context's value has been updated on step n is,

$$(7) \qquad \begin{aligned} & q(i,k_b'(i)) + d \sum_{j=i}^{N} p(i,j,k_b'(i))\ v'(j,n-1) \\ & + d \sum_{j=1}^{i-1} p(i,j,k_b'(i))\ v'(j,n) \end{aligned}$$

The value of context i under a decision $k_b'(i)$ after the (i-1)th context's value has been updated on step n+1 is given by (7) with n and n+1 replacing n-1 and n in the two sums. In the argument that follows $k_b'(i)$ will be the decision selected on one step of the leak-back process, for context i. The value of the context i at this step will be compared with the value of the context i at the next step under the same decision $k_b'(i)$. Taking the difference, D(i,n+1), to be the change in the value of the context i between the nth and the (n+1)th step under the decision $k_b'(i)$ selected at the nth step,

$$(8) \qquad D(i,n+1) = \begin{aligned} & q(i,k_b'(i)) - q(i,k_b'(i)) \\ & + d \sum_{j=1}^{N} p(i,j,k_b'(i))\ (v'(j,n) - v'(j,n-1)) \\ & + d \sum_{j=1}^{i-1} p(i,j,k_b'(i))\ (v'(j,n+1) - v'(j,n)) \end{aligned}$$

If the two sums in (8) are non-negative then D(i,n+1) will be non-negative.

Now, as I mentioned above, $v'(i,1) = 0$ for all i. So from (8)

$$(9) \qquad D(1,2) = d \sum_{j=1}^{N} p(1,j,k_b'(1))\ v'(j,1)$$
$$= 0$$

and

$$(10) \qquad D(2,2) = d\ p(2,1,k_b'(2))\ v'(1,2)$$

D(2,2) will be non-negative since,

(a) d, the discount factor, is non-negative,

(b) all p(i,j,k(i,x)), the transition probabilities, are non-negative,

(c) v'(1,2) is non-negative (because it is just $\max_{k(1,x)} q(1,k(1,x))$ from (4), since all the context values, v'(j,1), are zero, and rewards, q(i,k(i,x)), are assumed non-negative).

In general D(i,2) is just,

$$(11) \qquad d \sum_{j=1}^{i-1} p(i,j,k_b'(i))\ v'(j,2),$$

from (8) and since v'(i,1) = 0 for all i. Now if D(2,2) is non-negative then v'(2,2) will be non-negative since (a) v'(2,1) is zero, and (b) a value for context 2 of at least v'(2,1) + D(2,2) must be selected according to equation (4). Given that v'(2,2) is non-negative, then, from (11), D(3,2) will be non-negative. Thus, by the same argument, v'(3,2) will be non-negative, and so on up to v'(N,2).

Now,

$$(12) \qquad D(i,3) = d \sum_{j=i}^{N} p(i,j,k_b'(i))\ v'(j,2) + d \sum_{j=1}^{i-1} p(i,j,k_b'(i))\ (v'(j,3) - v'(j,2))$$

The first sum will always be non-negative; I have just shown that all D(j,2) are non-negative so that v'(j,2) is not less than v'(j,1) for all j.

D(1,3) will be non-negative since only the first sum occurs in D(1,3). This in turn means that v'(1,3) will not be less than v'(1,2) since the decision used for context 1 on step 2 gives no less a value on step three (because D(1,3) is non-negative).

Thus D(2,3) will be non-negative since both (i) the first sum in (12) is non-negative, and (ii) for D(2,3) the second sum in (12) is non-negative, v'(1,3) being not less than v'(1,2). Since D(2,3) is non-negative v'(2,3) will not be less than v'(2,2). This is because the decision used on step 2 gives at least the same value on step 3. The maximum value given by any particular decision will be at least v'(2,2). This in turn means that D(3,3) is non-negative and so on.

From (8) D(i,n+1) is non-negative if all of v'(j,n) are not less than v'(j,n-1) for $j \geq i$ and all of v'(j,n+1) are not less than v'(j,n) for $j < i$. If D(i,n+1) is non-negative then v'(i,n+1) will be at least v'(i,n). Therefore D(i+1,n+1) will in turn be non-negative, since all v'(j,n) are not less than v'(j,n-1) for $j \geq i$ and all v'(j,n+1) are not less than v'(j,n) for $j < i$. I have shown above that the first three iterations of D values are non-negative, so all the D values will be non-negative. Therefore the values of contexts can only increase or stay the same.

This concludes the first part of the proof; the values on contexts cannot be decreased by leak-back. [Remember the assumptions that the productions, transition probabilities and immediate rewards are not changing, that reward does not grow without limit, that transition probabilities add to one, and that rewards, q, are non-negative.]

I will now explain the third part; leak-back cannot converge to a sub-optimal policy. Suppose for a moment that it did converge to a

suboptimal policy k*. Leak-back would be performing the operation given by equation (13), since all the $k'_b(i)$ would not be changing.

$$v'(i,n+1) = q(i,k^*(i)) + d \sum_{j=i}^{N} p(i,j,k^*(i))\ v'(j,n) + d \sum_{j=1}^{i-1} p(i,j,k^*(i))\ v'(j,n+1) \tag{13}$$

This is a method of solving the set of linear equations given in equation (14), for the context values, v*(i).

$$v^*(i) = q(i,k^*(i)) + d \sum_{j=1}^{N} p(i,j,k^*(i))\ v^*(j) \tag{14}$$

Now, for leak-back to converge, as we have for the moment supposed, it must be true that

$$v(j,n+1) = v(j,n) \qquad \text{for all } j.$$

Thus (13) would reduce to (14). The context value estimates would be the true context values for that policy k*.

If k* is not the optimal policy then there will be a context for which some other decision will give a higher value (Howard, 1960). That is, once (13) converges to (14), if not before, leak-back will find a better policy, if there is one. Thus leak-back cannot converge on a sub-optimal policy.

So leak-back converges to the optimal values for contexts, since during leak-back: (1) context values cannot decrease; (2) context values cannot exceed the optimal values; and (3) context values cannot converge to non-optimal values.

The possibility of the leak-back "sitting on" a sub-optimal policy for a large number of iterations does not upset the decision process.

There is nothing special about the convergence of leak-back as far as the MCLS is concerned. Leak-back keeps on operating regardless of convergence. The MCLS goes on selecting and performing actions regardless of convergence. The leak-back process never stops, at least not over the lifetime of the robot.

VI.3 DISCOUNTING OF FUTURE REWARDS

The expected future values of actions may be discounted either (i) on the basis that expected future rewards may not be received, or (ii) on the basis that rewards received in the future are worth less than rewards received now. In (i) discounting may represent the possibility that some event or events will change the system that the leak-back process is optimizing. Thus sooner rewards are worth more than later ones because they are more likely to be obtained before the system changes.

Discounting of future rewards causes shorter paths to reward to be preferred to longer ones. However, discounting does not result in an optimally short path to reward. Since the discounting method penalises transitions unequally, it does not always find the shortest path. Consider the example shown in Figure VI-3. The decision process is in a state S. Given decision A, there is a probability of one that the next state will be S_n, a state that happens to be n steps away from the only rewarded state S_r, in a "direct line". I call a state "n steps

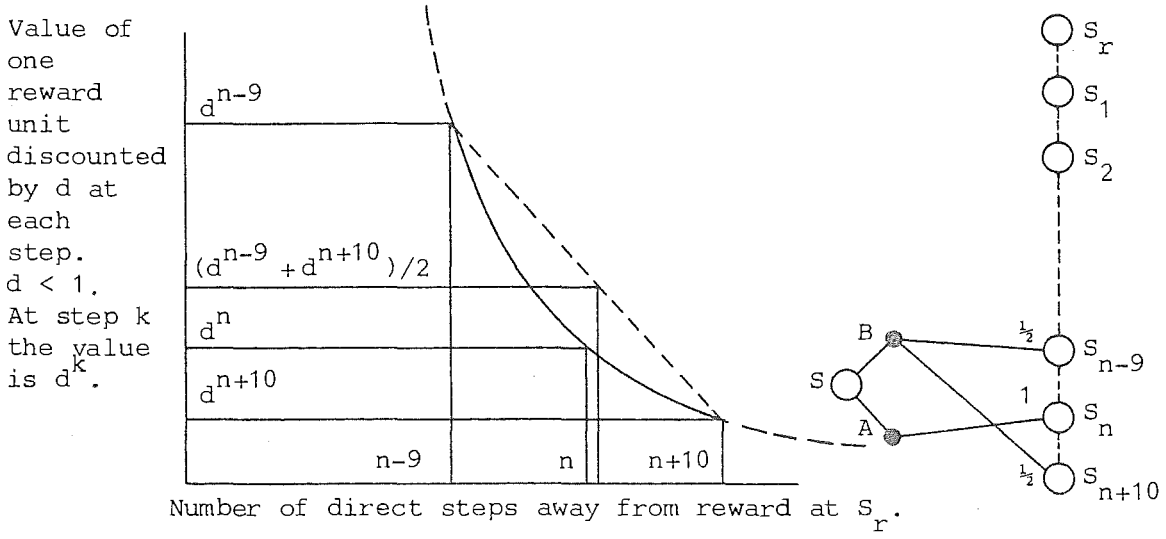


Figure VI-3 Discounting is suboptimal for minimizing path length. Discounting provides a suboptimal solution to the problem of finding the shortest path to reward. The value of decision A is d^n. The value of decision B is $(d^{n-9} + d^{n+10}) / 2$, which is greater than d^n, as shown in the Figure. So if discounting was employed, action B would be chosen in favour of action A. However, the optimal decision for the shortest path to S_r from state S is decision A. This provides an expected average path to reward of n steps. Decision B provides an expected average path length to reward of $n + \frac{1}{2}$ steps. In order to minimize the path length, transitions should be costed equally, as explained in the text. A "direct step" is a transition with a probability of 1, and only one decision. For example, once state s_{n+10} is entered, the only possible sequence of transitions is the series of $n + 10$ transitions to S_r.

away from S_r in a direct line" when it is a state from which S_r is reached with probability one in n steps. Once in S_n the state S_r will be reached in n transitions, each with probability one. Given decision B, there are two equiprobable next states. The two states are S_{n+10}, a state that is n+10 steps away from S_r in a direct line, and S_{n-9}, a state that is n-9 steps away from S_r in a direct line. Discount factors exist that will cause the decision B to be selected. As shown in Figure VI-3, B's estimated value may be greater than A's. However the decision A should be chosen in order to obtain, on average, the shortest path to reward. The average path length for the choice B is the average of n+10 and n-9, which is greater than n.

In order to select the optimal decision, a method should be used which applies the same penalty to each transition (see Howard, 1971). A cost would be subtracted from the value of each context j, when working out the value of context i. Future rewards would not be discounted. Under this scheme the value of B will be proportional to the average of n-9 and n+10. This is because the reward on any state in a direct line from S_r will be the reward on S_r, minus the cost per transition multiplied by the number of transitions. The curve in Figure VI-3 would be a straight line.

At the beginning of the chapter it was pointed out that future rewards must be discounted if leak-back is to solve for the maximum total expected future rewards. Then context values are bounded. Discounting is not required for leak-back to minimize the number of transitions made before reaching the context offering the highest immediate reward.[4] Following Howard (1971), the formal representation of the shortest path problem assumes any contexts with immediate reward to be followed by a single transition to an unrewarded "trapping" state. The transition to a trapping state has a probablity of one. The system will obtain reward at one of these immediate-reward contexts, and then

enter a state which offers no reward and from which it cannot escape.

Now, there may be states from which an immediate reward state cannot eventually be reached. The values of these states may go on decreasing, as leak-back progresses, if they are a loop of states. The costs of transitions go on decreasing the context values without the states ever getting reward. However, in practice this continual decrease of these context values poses no problems. If there is no way to get to any reward from a state then there is no solution to the shortest path problem for that state.

The build up of negative reward in a loop of states may leak-back to parts of the system from which reward can be reached. As the negative reward builds up on the states from which reward cannot be reached, paths from other states will be more and more cut off. Actions which may lead to a loop of states without reward will tend not to be chosen. Never ceasing build up of negative reward on states from which reward cannot be reached should not prevent the optimal decisions in other states being converged on by leak-back.

So the amount of reward that can be received is bounded, whenever there is actually a solution to the shortest path problem. In reality, once the reward is obtained, a new problem is set up. This new problem is to find the immediate-reward context that is closest to the immediate-reward context just reached. Finding a policy that maximises the total reward in the formal representation for the shortest path problem actually finds a policy that gives the shortest route to a rewarded context (Howard, 1960).

Andreae (1977a, 1982a) implements a scheme that is a combination of the discounted-maximum-total-reward solution and the undiscounted shortest-path-to-reward solution. He implements a shortest path method, but with discounting and no costs. Andreae's method has the advantage of the shortest path solution; if a context is rewarded then its value

can be set at a maximum level, and any reward leaking-back to it ignored. Reward past a context which has immediate reward is not important for the robot in finding a path to the closest rewarded context. This makes the leak-back process simpler because on a context, a, it need consider only the immediate reward value, q(a,k(a,x)), of an action or, if the immediate reward value of each action is zero, the future reward value, v(a), but never both values. It also means that the range of possible reward values is limited to the maximum immediate reward possible, rather than the much higher values which could accumulate if all discounted future rewards could be added to immediate reward.

Andreae (1977a; Andreae & Cleary, 1976; Andreae & Andreae, 1979) has used a form of reward called "novelty", which is stored with an MCLS production when it first occurs. This is a way for a learning system to (i) set its own goals, and (ii) have a reward system that is not completely built-in. Novelty will depend on the learning system's experience to a greater extent than built-in rewards because novelty is not specific to certain situations or events. Since the goal of the system is more to reach novel situations than maximise the novelty it obtains, it makes more sense to use a path length minimizing method of leak-back [see Andreae (1982a) for example].

The shortest-path-to-reward optimization method is suitable for a robot that becomes satiated. A satiated robot would not make decisions on the basis of rewards. Future rewards may not be important to a robot that has just received some reward. A satiated robot might make decisions on the basis of novelty rather than reward. Leak-back in a satiable robot should find the shortest path to reward, not optimize the total expected future rewards.

VI.4 LEAK-BACK IN A MULTIPLE CONTEXT

I have considered only a single context type or rule in the arguments above. In general an MCLS will have several, interacting rules. Actions are performed on the basis of the joint predictions of rules. Leak-back occurs only within each rule, not between them. I explain in this section how a second multiple context (MC) might enable goals in one rule to affect the predictions of other rules.

Andreae (1980b; 1982a) describes an MCLS that has two MC's in it. One, MC-H&N, operates in the present---the "Here and Now"---and the other, MC-T&T, operates at other "times and places"---in the "There and Then"---within the LTM of the MCLS. Andreae (1982b) describes another MCLS with two MCs in it; MC-H&N and MC-W&W (MC-Where & When). MC-W&W is very like MC-H&N. The differences are stated below. The MC-T&T and the MC-W&W obtain their stimuli from the predictions made by the MC. The MC-H&N obtains its stimuli from the environment. None of the actions of MC-T&T are performed by the robot. Of the actions selected by MC-W&W, only the speech actions are performed. The arm-robot's MC in chapter V is an MC-H&N.

Since the MC-T&T predicts its own stimuli and since its actions are not performed by the robot body, it can operate solely within the memory of the MCLS. MC-T&T follows paths of actions and stimuli within LTM, which may be followed later by MC-H&N. Thus the MC-T&T may be seen as the "planner" of the MCLS. It deals with the predicted future. When it doesn't perform speech actions, MC-W&W may follow paths of actions and stimuli within LTM, which MC-H&N may not be able to follow. MC-W&W may be seen as dealing with the "imagined" future.

The MC-T&T may leave "trails" in the memory for the MC-H&N to follow. As the MC-T&T goes ahead in LTM it may find reward or novelty goals, and bias the MC-H&N decisions toward the goals by leaving trails. The trails are left as markers on the productions the MC-T&T

followed to a goal. The MC-T&T described by Andreae (1980b) leaves trails only in the rules in which it finds goals. This is very similar to the leak-back process. Possible future goals in a rule's LTM affect the predictions of that rule. Both (i) leak-back from goals, and (ii) trails to goals, occur only within a rule's LTM. The MC-T&T described by Andreae (1982a) leaves trails in several rules whenever it reaches a goal in a rule. This way of having possible future goals affect the decisions of the MC-H&N complements the leak-back process. Leak-back of reward and novelty occurs within a rule. The MC-T&T enables future goals in one rule's memory to affect the MC-H&N's decisions in other rules. Novelty reached by MC-T&T in a context causes trails to be left in other types of context. Thus the MC-T&T enables future reward and novelty goals to be generalized to a multiple context from a single context.

The MC-T&T in the MCLS discussed by Andreae (1982a) is able to store productions in the MCLS's LTM. MC-T&T may store the actions that it selects as it moves through the memory of the MCLS. One effect of this is to enable MC-T&T to generalize "in the future" of MC-H&N. As discussed in section 2.3.2 of chapter IV, actions can be generalized from one situation to similar situations by an MC. Generalizing in the future of MC-H&N causes novelty goals to be created. The new productions stored by MC-T&T are marked as novel, just as all new productions are. In an MCLS with reward goals, the "future generalizing" of MC-T&T could create new paths to reward. The new productions stored by MC-T&T might be followed by rewarded productions. This would cause a transition to be formed through the new productions, allowing reward to leak-back to the productions that preceded the new ones.

Andreae (1980b pp.8-9) also suggests that the MC-T&T might "speak" to the the MC-H&N with "inner speech". When the MC-T&T selects a speech

action, a speech stimulus corresponding to that speech action is put into the MC-H&N. This would allow more direct communication from MC-T&T to MC-H&N than either the laying of trails or the storing of actions by MC-T&T. As Andreae mentions, suggestions of inner speech for the MC-T&T can only be speculative and vague at the moment. If we can devise a task which a robot cannot perform without having its MC-T&T use inner speech, then we will be able to say more.

Andreae (1982b) suggests that an MC-W&W may be needed if one particular MCLS is to answer questions in a reasonable way. MC-W&W's speech actions are performed; that is "spoken" aloud. So MC-W&W communicates with MC-H&N using speech. MC-W&W's speech actions go into MC-H&N's contexts. MC-W&W does not leave trails.

A second MC can enable goals in one rule to affect the predictions of other rules by (i) MC-T&T leaving trails in the memories of several rules whenever a goal is reached in one rule, (ii) MC-T&T creating novelty and transitions to reward by generalizing in the "There and Then", (iii) MC-T&T speaking to MC-H&N, by MC-T&T doing speech actions which are put into the contexts of MC-H&N, but not performed by the robot's speech apparatus, and (iv) MC-W&W's speech actions being "spoken", and put into MC-H&N's contexts.

VI.5 CONCLUSION

A single rule, or MDP, with leak-back can select optimal actions, as long as the productions, transitions, and rewards in the rule are not changing. Leak-back is a successive overrelaxation method, with an overrelaxation factor of 1. Expected future rewards are discounted. Discounting of future rewards does not always result in finding the shortest path to a goal.

An MC-T&T can "go ahead" in the LTM of an MCLS, biassing the actual decisions of the MCLS according to the rewards found in contexts. An MC-W&W can go ahead in the LTM of an MCLS, and "speak" to the MC-H&N from the "imagined" future.

NOTES FOR CHAPTER VI

1. Kemeny & Snell (1960) call what I, and Howard (1960) call an MDP, a Markov decision chain. By MDP they mean one in which the probabilities of transitions may be different at different stages of the process.

2. Pegg (1983a; 1983b) has described an efficient algorithm for implementing leak-back in an MCLS. One iteration of leak-back can be performed on a rule's LTM in a tenth of a second, in the MCLS in which the algorithm is used.

3. Recently, however, Hartley et al (1980) have suggested that "silicon chip technology" (p.vii) may lift the curse dimensionality that afflicts the solution of MDPs.

4. The proof of leak-back's convergence to an optimal policy for the discounted total future expected reward problem can be extended to the shortest path problem. The two differences are that (a) the rewards in the proof are assumed positive, whereas there are costs, or negative rewards, in the shortest path problem, and (b) the discount factor was assumed less than 1 in the proof, whereas there is no discounting in the shortest path problem; that is the discount factor is 1.

Now, the proof of leak-back's convergence did not itself rely on the discount factor being less than 1. Discounting was used only to guarantee that the context values did have bounded optimal values. However, in the shortest path formulation the context values are bounded as long as there are no loops of states from which reward cannot be reached. So it is not necessary to assume that the discount factor is less than 1.

Although it was assumed that immediate rewards were non-negative in the shortest path problem, leak-back must also converge when there are negative rewards, or costs, as long as the context values are bounded. Equation (4) rewritten in matrix form is,

$$(15) \quad V^{n+1} = \max_k \; Q(k) + dL(k)V^{n+1} + d(U(k) + D(k))V^n$$

where P may be written as $P = D + U + L$

D is a diagonal matrix, U an upper triangular matrix and L a lower triangular matrix

where V and Q are vector forms of v and q in (4), and P is a matrix form of p in (4).

In general leak-back will converge to optimal context values if and only if, for each policy, the iteration given by (15) for that policy, converges. Otherwise leak-back would diverge on some policy or policies. Now rewriting (15), for a fixed policy,

$$(16) \quad V^{n+1} = d(I - L)^{-1}(U + D)V^n + (I - L)^{-1}Q$$

(16) has the form,

$$(17) \quad V^{n+1} = GV^n + B,$$

which converges if and only if the spectral radius of G, S(G), is less than 1 (Theorem 1.4. Varga, 1962). The spectral radius of a matrix is the absolute value of whichever eigenvalue of that matrix has the largest absolute value.

Now since (15) converges, as shown in section 2.2, then S(G) must be less than 1. Therefore leak-back converges for negative reward values, as well as non-negative ones, since G is independent of Q. This is true for d equal to 1, as pointed out above. Therefore leak-back can solve the shortest path problem, represented by (15) with (a) d equal to 1, (b) rewards negative or non-negative, and (c) as long as there are no loops of states from which positive rewards, and the "trapping" state, cannot be reached.

CHAPTER VII

A FRESH LOOK AT CONDITIONED REFLEXES

Suppose a robot has a movable eye. This might be necessary so that the robot could look over a wide visual angle with a narrow, detailed vision system. Detailed visual information over a wide angle will be needed for some robot tasks; for example, so that a robot can put a peg in a hole whose location is unknown.

I said at the start of chapter II that in order to get a robot to perform a movement task, a teacher may (i) tell or show the robot the motor commands, or actions, to send to its body, (ii) tell or show the robot the movements and forces to achieve, or (iii) give the robot a task description in more complex terms than the movements and forces. However, a teacher will have no natural feeling for the eye actions or eye movements required for a task. Humans are not nearly as aware of their own eye movements as they are of their own arm movements. As pointed out in chapter II, no existing robot can perform complex movement tasks in the real world, given the sorts of task description we are used to giving. Even if such a robot did exist, a teacher would find it difficult to give the robot a description of the eye movement part of the task. He would not have the same natural feeling for the visual information required, as he would for the physical manipulations required.

Instead of the teacher providing them, the eye movements might be preprogrammed to occur in a specific way, when the robot is built. For example, a robot's eyes might be preprogrammed to follow its hand.

However, it may be very difficult, perhaps impossible, to preprogram a robot to automatically perform the required eye movement sequences. Sequences would be required that enable the robot to sense the visual

information appropriate for all the tasks it may perform.

So a robot may need to learn to use preprogrammed eye movements in situations for which they are not preprogrammed.

In this chapter I show that a robot with one of the simplest forms of preprogrammed eye movement can learn to perform eye movements in situations for which the eye movements are not preprogrammed. Eye actions are preprogrammed to occur when a particular stimulus occurs. The actions are called "reflex-actions". The stimuli are called "reflex-triggers".

A head movement system may be required, as well as an eye movement system, for a robot that has a moving eye. Then (a) other sensory devices, for example hearing devices, may be pointed in directions other than the "looking" direction, and (b) the eye movement system can be small and therefore move very rapidly.

In this chapter I suggest a design for a head movement system. The head movement system is taught using the leading method. While in the training mode the head movement system resists external forces, so that the head is not floppy. Thus, in contrast to the training mode that arms have, as discussed in chapters II, III, IV and V, the head's training mode is not passive. The stability and feasibility of a control system for a resistive training mode, are established in this chapter.

In section 1 I explain how a robot can learn to perform reflex actions. An example is given in the appendix. In section 2 the head movement system for a real robot is proposed. In section 3 reflexes and leading are compared. Briefly: leading is suitable for arm movements; reflexes are suitable for eye movements; and "back-reflexes", a variation of reflexes, are suitable for speech.

VII.1 REFLEXES

There is one basic principle involved in the learning of reflexes described in this chapter. Firstly, an association is formed between a reflex-action and the situation in which it is triggered. Secondly, that association is recalled when a similar situation occurs, but in which the reflex-action is not triggered. For example, Figure VII-1 shows the performance of a look-left reflex action, which results in a look-left reflex movement. A visual stimulus caused by a movement on

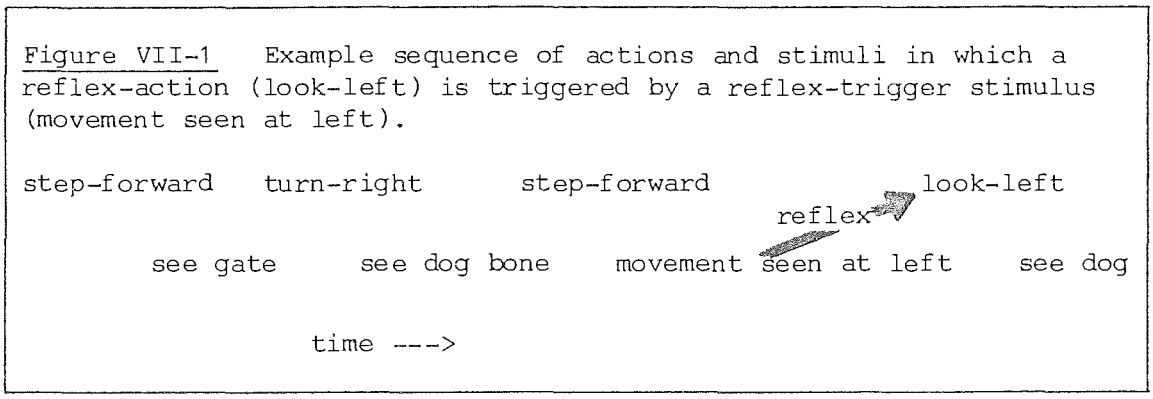


Figure VII-1 Example sequence of actions and stimuli in which a reflex-action (look-left) is triggered by a reflex-trigger stimulus (movement seen at left).

the left side of the robot, triggers the look-left reflex-action. Suppose the robot remembers that look-left occurred, not straight after "see dog bone", but after the stimulus that followed "see dog bone". An association is formed between "see dog bone" and look-left. I shall write this association as:

2nd latest stimulus = "see dog bone" ---> look-left.

Figure VII-2 shows a similar situation to the one before the reflex-action look-left is performed in Figure VII-1. The only

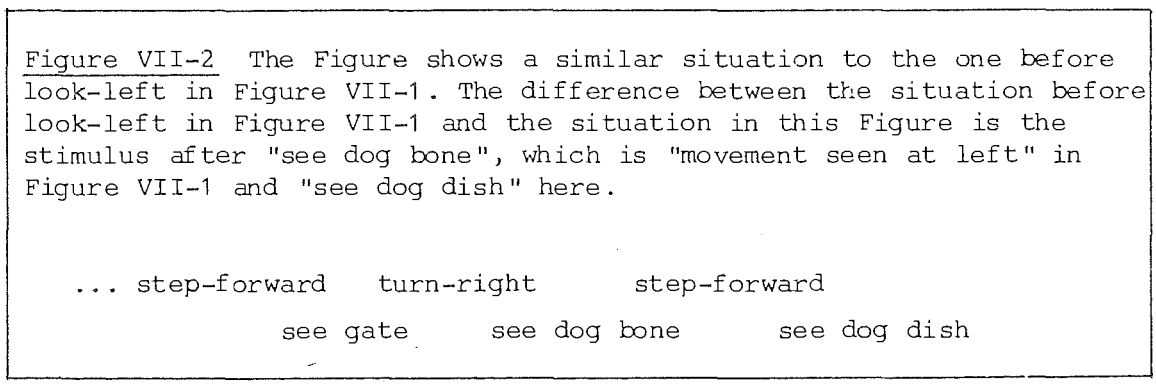


Figure VII-2 The Figure shows a similar situation to the one before look-left in Figure VII-1. The difference between the situation before look-left in Figure VII-1 and the situation in this Figure is the stimulus after "see dog bone", which is "movement seen at left" in Figure VII-1 and "see dog dish" here.

difference is that in Figure VII-2 the stimulus after "see dog bone" is the visual stimulus of seeing the dog dish, rather than the reflex-trigger caused by movement in the left of the visual field.

Suppose that after "see dog dish" in Figure VII-2, the robot recalls the previously formed association,

2nd latest stimulus = "see dog bone" ---> look-left.

Look-left may be performed as a result of this association being recalled.

I shall compare this forming and recalling of associations, the basic principle of this chapter's reflex learning, with the paradigms of classical and operant conditioning in section 1.2. The notion of "association" will be dealt with more precisely in sections 1.3 and 1.4.

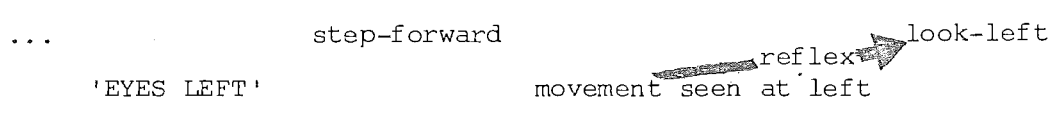
Note that when the association with the second latest stimulus causes the reflex-action to be produced, direct associations with other stimuli may be formed. For example, once look-left has been performed after "see dog dish" in Figure VII-2, as association

latest stimulus = "see dog dish" ---> look-left

may be formed. Then the stimulus "see dog dish" may elicit the action look-left, even though "see dog dish" is not a reflex-trigger for look-left. When the associations with these other stimuli cause the reflex-action to be produced, more associations may be formed. Those associations may in turn cause the reflex-action to be produced, and so on. The forming and recalling of associations can enable reflex actions to be learned in many situations.

Suppose that instead of the sequence in Figure VII-1 the sequence in Figure VII-3 occurs. The teacher says 'EYES LEFT' just before a look-left reflex occurs.

Figure VII-3 Speech followed by a reflex



'EYES LEFT' is a speech stimulus. Now, if the robot employs the association scheme suggested above---the 2nd latest stimulus before a reflex-action being associated with the reflex-action---then an association

2nd latest stimulus = 'EYES LEFT' ---> look-left

will be formed when the sequence in Figure VII-3 occurs. The teacher could elicit the action look-left from the robot in any situation by saying 'EYES LEFT'. After the next stimulus, the robot would recall the association 2nd latest stimulus = 'EYES LEFT' ---> look-left, and perform look-left.

It will be explained that a multiple context learning system (MCLS) can be designed to form and recall this sort of association between reflex-actions and speech or other stimuli. However, this alone does not mean that the learning of reflexes will be of any practical use to a robot. For example, reflexes might be "overgeneralized"; they might be learned and performed in so many situations as to be a hindrance to teaching. To establish that reflexes may be of some practical use, I show that a particular MCLS, the 1979 version of PURR-PUSS (Andreae, 1979b), can learn to perform a reflex-action without the reflex-trigger occurring. As pointed out in this chapter, PURR-PUSS generalizes reflex actions with more difficulty than she generalizes other actions. Also, PURR-PUSS selects an action that minimizes generalization. PURR-PUSS, a well-tested MCLS, has been taught many tasks in which the teaching was not hindered by overgeneralization (see Andreae, 1977a). I was able to show reflexes being learned quite generally, without modifying the characteristics of the 1979 PURR-PUSS. Since 1979 other versions of PURR-PUSS have been produced (Andreae, 1980a; 1980b; 1981; 1982a). In this chapter only the 1979 one will be referred to as "PURR-PUSS".

Briefly, for an action to be performed by PURR-PUSS there must be three separate associations recalled for that action. For the example performance of a look-left reflex-action by PURR-PUSS the three associations given in Figure VII-4 are formed and then later recalled.

1. 370 070 310 300 220 100 340 ---> 200
2. forward left left forward forward right forward ---> look-left
3. look-left/200 look-centre/40 look-right/10 ---> look-left/200

Figure VII-4 The three associations formed and recalled by PURR-PUSS for the performance of a look-left action in the illustrative interaction reported in the appendix. The look-left action is "*B" in the illustrative interaction. The numbers in the Figure are visual stimuli from PURR-PUSS's simulated environment. "forward", "left" and "right" are actions which move PURR-PUSS around the environment. The three "look" actions are eye movements enabling PURR-PUSS to look in other directions than the one she is facing. Detailed explanation is given in the appendix.

In PURR-PUSS a look-left action is triggered by the visual stimulus of a dog to left of the robot. A dog to the left is represented by a 4 in the units digit of a visual stimulus number. For example, as set out in the next paragraph, 344 triggers a look-left, but 340 does not.

Association 2 in Figure VII-4 is formed when look-left is triggered after the performance of the seven actions on the left side of that association. Association 3 is formed when the look-left is triggered as shown below

look-left ... look-centre ... look-right ... ... look-left

reflex

200 ... 40 ... 10 ... 344 200

where there are only non-look actions between the look actions. Association 1 cannot be formed when look-left occurs because 340, the last stimulus before 200, does not trigger look-left. Instead

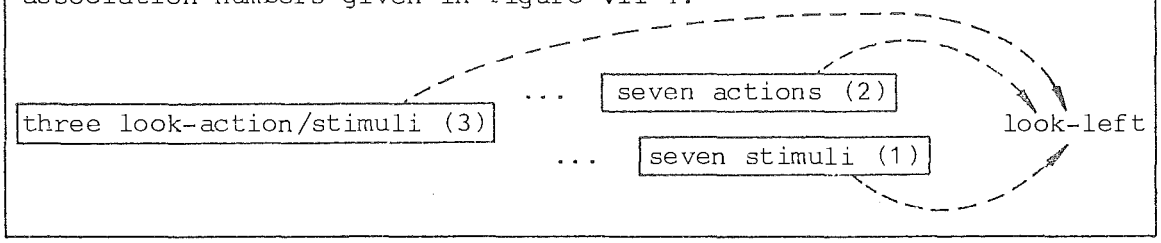
association 1 is formed when this sequence occurs

forward	left	left	forward	forward	right	forward	right
	370	070	310	300	220	100	340	200

The visual stimulus 200 occurs after "right", as well as after "look-left".

When all three associations are recalled together, the look-left action is performed, even though the reflex-trigger 344 did not occur. The three associations are recalled together once the seven actions on the left side of association 2 and the seven stimuli on the left side of association 1, follow the three look-action/stimulus pairs on the left side of association 3. This is depicted in Figure VII-5. As I explain later in the chapter, association 3 keeps "predicting" the look-left until associations 1 and 2 are recalled and agree with it. Then look-left is performed.

Figure VII-5 The three associations given in the text agree to "predict" the action look-left. The numbers in brackets are the association numbers given in Figure VII-4.



Section 1.1 explains that the methods of leading, guiding and programming-by-the-teacher may not be suitable for teaching eye movement reflexes to a robot. In section 1.2 "classical" and "operant" conditioning paradigms are compared to this chapter's reflex learning. Section 1.3 briefly explains both (i) how reflexes can be implemented with MCLSs, and (ii) how reflex actions can be learned by MCLSs. Section 1.4 summarizes the section of the appendix which both (i) describes reflexes in PURR-PUSS, and (ii) gives an illustrative

interaction of PURR-PUSS learning reflexes. The appendix also explains MCLSs' reflex learning, in more general terms than the learning of look-left by PURR-PUSS.

VII.1.1 Reasons for preprogramming eye movements

Eye movements may need to be preprogrammed into a robot with movable eyes because the three other methods of putting them in---(a) the leading method, (b) the guiding method, and (c) programming by the teacher---may not be suitable.

The leading method is not suitable because it would be difficult for a teacher to teach a robot to look around, by physically moving its eyes around.

The guiding method may not be suitable because a teacher may find it difficult to guide the robot's eyes through the movements for looking around. For example, unless the teacher looks through the robot's eyes and the robot's eye movements are "slaved" to the teacher's eye movements, the teacher will not be using his own natural ability at moving his eyes. He would normally be unable to use his own hand-eye coordination skills in guiding a robot's eye, as he can when guiding a robot's hand.

Programming-by-the-teacher is not suitable because the teacher would find it difficult to select the eye movements a robot should perform during a task. A teacher's awareness of the hand movements he performs during a task may help him in programming that task into a robot. However, humans have very little awareness of the eye movements they make during a task.

I am trying to design robots that are more adaptable than existing robots. To be adaptable, a robot with a moving eye must be able to use eye motor commands that are preprogrammed to occur in particular situations, in other situations.[1]

I show in this chapter that a robot with an MCLS in it can learn to perform a preprogrammed, reflex movement, even when the reflex triggering stimulus is not present.

VII.1.2 A Natural Way

There is evidence of "prepared (inborn) synaptic connections" in humans and animals, and even that the "equipment" that receives stimuli and that which performs reflex actions matures ahead, in time, of other equipment (Anokhin, 1974).

I am able to say quite specific things about reflexes in the MCLS PURR-PUSS. In contrast, two other stimulus-action, or "stimulus-response", paradigms of learning are not so precisely prescribed. The two are "classical conditioning" and "operant conditioning".

The general learning of reflexes that I describe in this chapter is similar to the formation of the classical conditioned reflex and to higher order classical conditioning. A certain stimulus, US, causes a built-in response R from an organism (see Rachlin, 1976; McGuigan & Lumsden, 1973). US is a reflex-trigger stimulus. R is a reflex-action. Another stimulus, CS, has no effect on the occurrence of R. CS may be "conditioned" to cause R by repeated occurrence of CS-before-US-causing-R. Soon CS will cause R without US being present. In this chapter situations, rather than individual stimuli, or CSs, are conditioned to produce reflex-actions. The situation could be the occurrence of a single stimulus, or a group of action-stimuli. Higher order classical conditioning involves the use of a conditioned CS to condition still other stimuli to cause R (Rescorla, 1973). However, this classical conditioning paradigm is deceptively simple. There are complications associated with the (a) contiguity of stimuli, (b) the repetition of stimuli, and (c) the reinforcement of the response (Hebb, 1958; Rachlin, 1976; Saltz, 1973; Sutton & Barto, 1981;

Grossberg, 1982). I am able to investigate reflexes in the specific framework of the MCLS. I must be precise and unambiguous, aiming for an actual robot implementation.

The second conditioning paradigm is that of operant conditioning (Skinner, 1938). An organism is put into an environment where it is "likely" to perform some desired---desired by the experimenter---action. If it does so, it is rewarded---"reinforced"---and the organism becomes "conditioned" to perform this action after whatever preceded its first occurrence. This would appear to require the spontaneous, or irrespective-of-situation, production of actions. Once the actions have been produced in this way in some situation, they are conditioned to occur following that situation.

In their work, John Andreae and his co-workers (Andreae, 1972-1983; MacDonald & Andreae, 1981; Andreae & Cleary, 1976; Andreae & Andreae 1978) have found it is necessary to prescribe precisely, through a multiple context, the conditions under which actions are learned. This precise prescription does not seem to be made in current conditioning theories and experiments, as exemplified by the collection of contributions in "Contemporary Approaches to Conditioning and Learning" by F.J.McGuigan and D.B.Lumsden (1973).

VII.1.3 Reflexes in MCLSs

VII.1.3.1 Reflex implementation A robot with an MCLS in it can be preprogrammed to perform movements by reflex; a certain action is performed when a certain stimulus is provided by the environment, as shown in Figure VII-6 [see also Andreae (1980a; 1980b; 1982a) for MCLS reflex implementation.]. The Figure also depicts "back-reflex", guiding, leading, spontaneous action production, and action selection by PURR-PUSS. Back-reflex is where there are predetermined action/stimulus pairs such that when one of the stimuli occurs the action of the pair is remembered as though it had preceded the

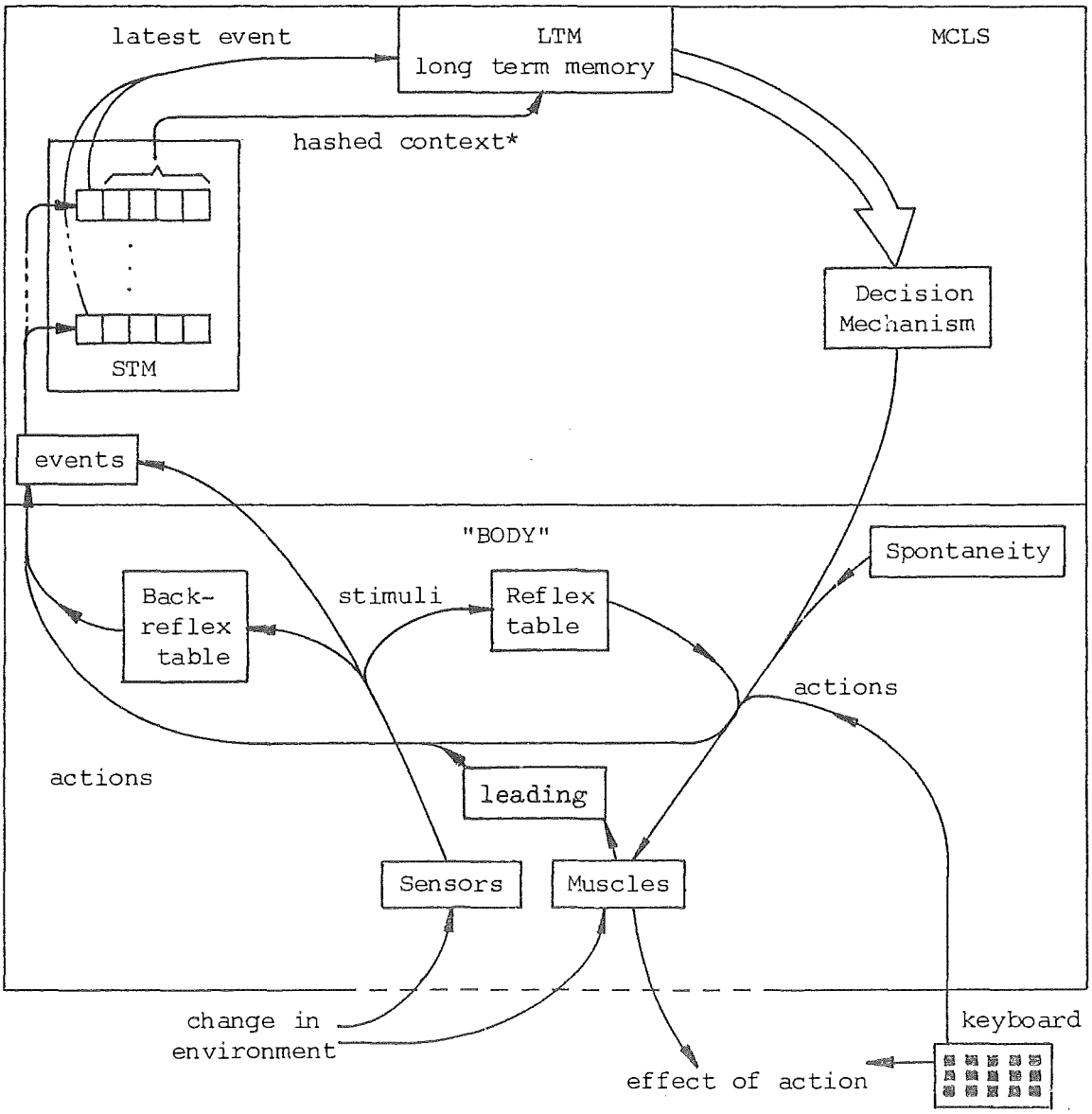


Figure VII-6 Ways for an MCLS to learn actions:

(a) If an action is recorded in LTM as having occurred after the current contexts on previous occasions, then the decision mechanism will send this action to the body, so long as enough contexts predict it. This is explained in section 2 of chapter IV.

(b) Actions may be given to the body spontaneously.

(c) The environment may move the muscles, causing a "led" action to be stored, as explained in chapters III and IV.

(d) If a stimulus matches one of those in the back-reflex table of stimulus-action pairs, then the associated action is remembered as though it had occurred before the stimulus, but it is not performed.

(e) If a stimulus matches one of those in the reflex table of stimulus-action pairs, then the associated action is sent to the body.

(f) The teacher may guide the MCLS via a keyboard.

*A hashing technique is used for storing and retrieving the predictions of PURR-PUSSs' contexts (Andreae, 1977a).

stimulus, even though it didn't. Mimic-speech in an MCLS (Andreae, 1977a) is an example of back-reflex. A teacher can "say"[2] a word, "Hello" for example, and the robot remembers an action for saying "Hello". The action is remembered as though it was followed by the stimulus "Hello". Back-reflex and ordinary reflex are compared in section 3, where back-reflex is explained in more detail.

An action performed by reflex can be remembered in the contexts in which it occurs; productions are stored which predict the reflex-action. The productions are associations between the context and the reflex-action. The operation of MCLSs was described in section 2 of chapter IV. Briefly revising from that section, the decision mechanism, indicated in Figure VII-6, looks at the productions in long term memory (LTM) to find the events that each current context predicts. An action will be performed if enough of the current contexts predict it. The current contexts are stored in short term memory (STM), as explained in chapter IV and indicated in Figure VII-6. An MCLS implements the forming and recalling of associations---the basic principle of this chapter's reflex learning---by storing productions in LTM and predicting actions with the current contexts in STM.

I will illustrate the actions that an MCLS performs, and the stimuli that it receives, by an Event Sequence diagram, or Event diagram for short. Actions and stimuli are put in two lines, but the time relationship between them is preserved by staggering them. An example, shown in Figure VII-7, is taken from the interaction in Figure VII-16, which is explained in the appendix.

In Figure VII-7, all actions and stimuli are specific "constants". If I wish to consider a more general type of event sequence, then I will use "variables" indicated by triangular brackets. When I write <action> or <stimulus> I have placed no restriction on what action or stimulus these represent. To indicate a particular constant, without

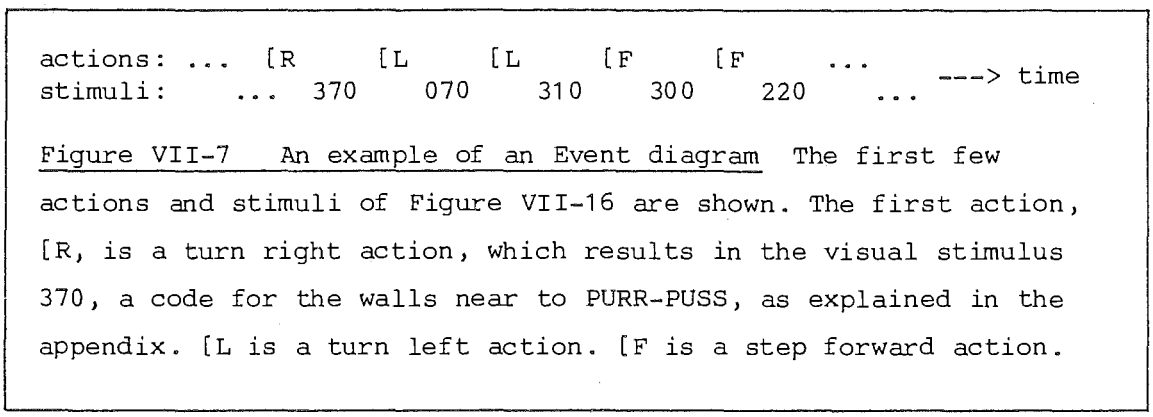


Figure VII-7 An example of an Event diagram The first few actions and stimuli of Figure VII-16 are shown. The first action, [R, is a turn right action, which results in the visual stimulus 370, a code for the walls near to PURR-PUSS, as explained in the appendix. [L is a turn left action. [F is a step forward action.

saying what the constant is, I will place a subscript outside the triangular brackets. Thus, whenever <reflex-action>$_a$ is written, I mean the same reflex-action.

VII.1.3.2 Reflex performance When a reflex-action occurs because it was triggered by a reflex-trigger stimulus, then a heavy arrow indicates the connection, as illustrated by the Event diagram of Figure VII-8.

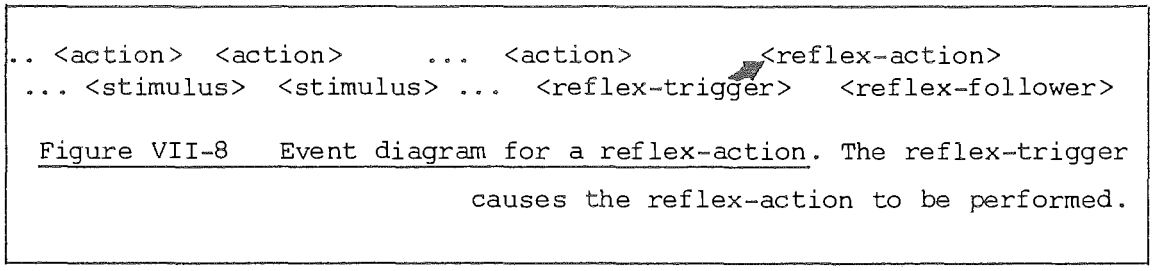


Figure VII-8 Event diagram for a reflex-action. The reflex-trigger causes the reflex-action to be performed.

Figure VII-9 shows the performance of a reflex-action that is not caused by the reflex-trigger. The stimulus before the reflex-action is not a reflex-trigger stimulus. Such a sequence of events can occur only if enough contexts predict the reflex-action/reflex-stimulus pair for the reflex-action to be selected and performed by the MCLS.

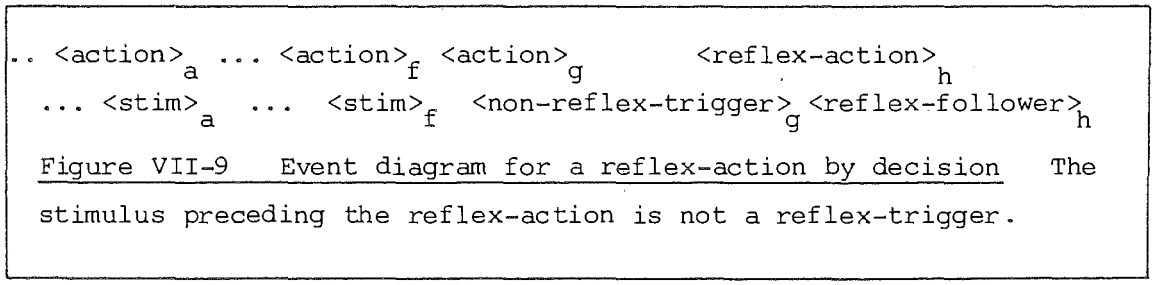


Figure VII-9 Event diagram for a reflex-action by decision The stimulus preceding the reflex-action is not a reflex-trigger.

As explained at the start of section 1, the main principle of reflex learning in this chapter is the forming of an association when a reflex action is triggered (Figure VII-8) and the recalling of the association to cause the reflex-action without a reflex-trigger (Figure VII-9).

There is only one difference between

(i) the event sequence in Figure VII-9, where a reflex-action is performed because of an MCLS decision, and

(ii) the event sequence in Figure VII-8, where a reflex-action is triggered by the reflex-trigger,

The difference is that in Figure VII-8 the stimulus before the reflex-action is the reflex-trigger, while in Figure VII-9 the stimulus before the reflex-action is not a reflex-trigger.

As a result of this difference, there are constraints on how productions may be set up to predict a reflex-action/reflex-follower pair. There are two constraints on the setting up of productions that predict a reflex-action. There are three contraints on the setting up of productions that predict a reflex-follower. The constraints are briefly described below.

Until a reflex-action has been performed by decision of the MCLS, as shown in Figure VII-9, all reflex-action performances will be ones that have been triggered by reflex-triggers, as shown in Figure VII-8. Therefore, any productions that predict the reflex-action in Figure VII-9 must have been formed when the reflex-action was triggered by a reflex-trigger stimulus, as shown in Figure VII-8. So for a production to predict the reflex-action in Figure VII-9, (a) it must not have had the reflex-trigger stimulus in its context when it was formed in Figure VII-8, and (b) it must not have the non-reflex-trigger in it in Figure VII-9. These are the two constraints on setting up productions that predict a reflex-action. The appendix explains how both constraints are met by PURR-PUSS. Two productions are formed in

PURR-PUSS to predict the reflex-action.

A production can be formed to predict the reflex-follower stimulus in two ways. Firstly, such a production could be formed as a result of the sequence of events shown in Figure VII-8. Then the two constraints, that the context of the production has neither the trigger stimulus in it in Figure VII-8, nor the non-reflex-trigger stimulus in it in Figure VII-9, must be satisfied for the reflex-follower to be predicted in Figure VII-9. The appendix explains how both constraints are met by PURR-PUSS, enabling the formation of one production to predict the reflex-follower.

Secondly, a production predicting the reflex-follower could be formed without the reflex-action occurring. The first two contraints need not be met, so long as a third constraint is met; the reflex-action must not precede the reflex-follower. The reflex-follower might occur after actions other than the reflex-action. Then a production predicting it could be formed without the reflex-action, or the reflex-trigger, occurring. The appendix explains how this constraint is met by PURR-PUSS, enabling the formation of a second production to predict the reflex-follower.

Now, it is quite simple to design an MCLS that will produce the sequence shown in Figure VII-9. With an appropriate selection of production templates and priorities a reflex-action could be predicted and performed in Figure VII-9 by productions which are formed in Figure VII-8. There must be a certain number of contexts predicting the reflex-action without the reflex-trigger being in the contexts. There must be enough contexts for the prediction of the reflex-action to be a higher priority than other predictions. If general learning of a reflex-action is required then enough of the contexts that predict the reflex-action must not be so specific to the situation in which the reflex occurs that they cannot predict the reflex-action in other

situations. For example, as well as not having the reflex-trigger in them the contexts must not have events in them which occur only when the reflex-trigger occurs. Otherwise they could not predict the reflex-action in the situations where the reflex-trigger did not occur. The various ways in which an MCLS can generalize actions from one situation to another, different situation are discussed in section 2.3.2 of chapter IV and in MacDonald (1982a).

A high priority speech context could be used to cause reflex-actions to occur in situations without the reflex-trigger, in the way suggested at the start of section 1. The teacher might say a sound just before the reflex-trigger triggered the reflex-action. For example, he might say 'eyes left' just before a look-left reflex eye action occurs. The reflex-action could then be stored in a production with that sound as the context; for example "'eyes left' ---> look-left". The teacher could teach the robot to perform the reflex-action in a situation by saying the sound, in the same way that he verbally corrects a led robot, as discussed in chapter III. He could say 'look left', causing a look-left eye movement. This verbal eliciting of eye movements is a verbal guiding method for eyes. I pointed out at the start of this chapter that a teacher would find it difficult to guide a robot's eye movements for a task. The point is just that an MCLS can easily be designed so that in principle reflex actions can be performed in non-reflex situations.

Since it is so easy in principle for an MCLS to learn reflexes, it is important to show that reflexes can be learned generally in practical situations, as pointed out at the start of section 1. In the appendix I show that the well tested MCLS PURR-PUSS can learn reflexes in its "SQUARES" environment. Section 1.4 summarizes that part of the appendix.

VII.1.4 Reflexes in PURR-PUSS

The appendix gives both the production templates for the rules in PURR-PUSS, and the decision procedure of PURR-PUSS. The appendix also explains just how a reflex-action may be produced by PURR-PUSS when the reflex-trigger does not occur to trigger it. Briefly, only three of PURR-PUSS's rules can predict a reflex-action, because of the constraints given in section 1.3.2. Prediction by the three rules is just enough to cause the reflex-action to be performed.

An illustrative interaction of PURR-PUSS learning a reflex look-left action is given in the appendix. The three associations given in Figure VII-4 are the three productions that predict the reflex-action/reflex-follower pair, look-left/200, in the interaction. The three productions are formed in the three rules mentioned above; one in each rule. The look-left action is shown as *B in the appendix. Briefly, the three productions are formed by the robot three times following a particular path in its environment. The first time along the path association 1 is formed, although no reflex occurs. In association 1 a context of seven stimuli is associated with the stimulus 200. The second time along the path the sight of a "dog", not there the first time, causes a reflex look-left action to occur. The dog is to the left of robot. Association 2 is formed; a context of seven actions being associated with the look-left action. As the robot comes back to the start of the path, reflex look-centre and look-right actions are triggered by the sight of the dog. The third time the robot goes along the path it does a second look-left action by reflex. This causes look-left/200 to be associated with the three previous reflex look-action/stimulus pairs. That is, association 3 of Figure VII-4 is formed. The fourth time along the path, the look-left action is performed by PURR-PUSS. The "dog" is not there to trigger the reflex.

In the appendix I explain how an association like association 1 of Figure VII-4 can be formed in almost any situation in the robot's environment. Briefly, an association can be formed between the stimulus 200 and a sequence of stimuli that are not related to the robot's immediate environment. For example, 200 might be associated with seven speech stimuli, instead of seven visual stimuli. The association can then be recalled in almost any environmental situation. Either the robot, or someone else must make the seven speech sounds. 200 would be predicted. That prediction might contribute to the performance of look-left. Association 1, which is used in the illustrative interaction, can be formed only in some environmental situations, since it represents a certain sequence of visual stimluli.

Also in the appendix the constraints on PURR-PUSS's learning of reflex-actions are investigated. Briefly, (i) in some situations three "dummy" actions must be performed before a reflex-action, if that reflex-action is to be learned in that situation, and (ii) all of a robot's actions cannot be reflex actions, if the MCLS PURR-PUSS is in the robot.

VII.2 MOVEMENT CONTROL FOR A ROBOT'S EYE AND HEAD

This section proposes a robot head movement system that employs a variation of the leading method.

Real learning robots are likely to have vision systems, as indicated in section 4.3 of chapter II. If a vision system collects detailed information in a small angular visual field only, as the human eye does[3], then the robot's eye must be able to look in different directions in order to collect visual information about all of the environment around it.

I pointed out at the beginning of this chapter that a head movement system may be required on a robot with an eye that moves. In spite of

the redundancy of movement, (a) the robot's visual sensor, its eye, may need to move about much more quickly than a head-sized component could move about, and (b) a moving head will be needed for separate reasons, such as orienting sound sensors. A system for the control of rapid eye shift movements, or saccades, is suggested in MacDonald (1979). Eye movements may occur by reflex.

My suggested head movement system employs the leading method. It could also respond to auditory reflexes, but I shall discuss only the leading method of teaching head movements.

The head movement sequences a teacher must teach a robot are unlikely to be as complicated as the arm movement sequences he must teach. The head movements needed in a movement task will not be as complicated as the arm movements needed, because (a) a head rarely has to apply forces to objects and manipulate objects, in the way that an arm does, and (b) a robot head will not have more than three degrees of freedom, and perhaps only one, whereas a robot arm will have five, six or more degrees of freedom. Therefore, since leading works for the more difficult teaching of arms, it should work for heads. Verbal correcting, discussed in chapter III, should work for heads too.

In the appendix I propose a slightly different leading arrangement to that in chapters III, IV and V. I propose a system that resists external movements even when it is in its training mode. There are a number of reasons why a resisting system might be useful, so it is worth considering. I have already shown how leading works with a completely relaxed control system during teaching, in chapters III, IV and V. Resistance to external forces during the training mode could make the teaching of arm movement tasks quite difficult for a teacher. Resistance should not make it so difficult for a teacher to teach head movements though, since the head movement sequences are likely to be less complicated than arm movement sequences. A floppy head might be

hard for a robot to control while it did other things, for example moving other parts of its body. So a resisting training mode may be some advantage.

My intention here is to demonstrate how reflexes and leading might be arranged in similar ways in a robot. Systems for muscle control in humans afford many ideas. In the appendix at the end of this thesis some comments are made about both the human visual system and muscle control. However, it is primarily important that the system be suitable for a robot. It is also important that I wish to give a robot only the bare essentials; then I really know what is necessary.

I consider movement about one axis only; the vertical axis of the robot's body. The appendix gives three realizations of the single axis head movement system; a servomotor control system, a biological analogue of the servomotor system and an analogue computer simulation. During the training mode, all three systems provide short term error feedback of head position. So over short periods of time, head position are resisted. Over a longer period of time, feedback decreases. The reference angle for the feedback c made to gradually approach the actual head position, causi to decrease. Once the reference position reaches the actua position, there is no error and therefore no resistance. I the appendix, laboratory results show the control system to be stable and feasible. The control system investigated in the laboratory took about fifteen seconds to almost completely stop resisting a disturbance to the head position.

VII.3 REFLEXES VERSUS LEADING

In section 3.1 reflexes, back-reflexes and leading are compared. It is explained that leading is suitable for arm movements; reflexes for eye movements; and back-reflexes for speech apparatus movements. In section 3.2 a similarity is pointed out in the conditions under which a reflex, back-reflex or led action may occur.

VII.3.1 Reflex, back-reflex and leading

Two forms of reflex have been discussed and implemented with MCLSs. They are the reflex discussed in this chapter, and the back-reflex, which has been called mimic-speech when used with speech actions (Andreae, 1977a).

The normal reflex mechanism operates by automatically causing a particular motor command to be generated whenever a particular sensory input occurs. The normal form of the reflex is not appropriate for certain sensory-input/reflex-movement pairs. For example, imagine that the speech stimulus "AR" causes a speech motor action which makes an "AR" sound with the speech apparatus. The robot would go on and on saying "AR" to itself. Imagine a reflex which caused a joint of an arm to rotate by positive ten degrees whenever a proprioceptive stimulus signalled a movement of that joint by positive ten degrees. Once moved by positive ten degrees, the joint would go on rotating until it reached its positive limit. These two sorts of sensory-input/motor-output pair are inappropriate for reflexes because they would nearly always cause the reflex mechanism to go into a loop.

The method of back-reflex has been implemented in an MCLS in order to enable it to acquire speech motor actions (see Andreae, 1977a). Back-reflex differs from the ordinary reflex in two ways. Firstly, when a back-reflex occurs, the back-reflex action is not performed, only remembered. Secondly, the action is remembered as though it occurred before the triggering stimulus. So, if someone were to say "AR" to the

robot then the motor command for "AR" would be remembered as though that motor command had occurred and then the stimulus "AR" had been heard. This is the normal sequence of events that occurs when the robot says "AR" itself. Speech back-reflexes enable a robot to learn to make speech sounds.

The teaching method of leading a robot through movements is similar to both reflex and back-reflex. An action is remembered automatically when the proprioceptive stimulus corresponding to that action occurs, as long as the robot arm is relaxed. For example, in chapter V it was explained that the arm-robot's arm can be moved up 40 degrees while the arm is relaxed. This causes the arm-robot to remember a +40 action. The +40 action is remembered as though it occurred before the stimulus for the 40 degree arm angle change. It is as though the +40 action had been performed, causing a 40 degree movement. The action is not performed, only remembered, just as back-reflex actions are not performed, but only remembered. The led action is remembered as though it happened before the stimulus, which is the normal sequence of events when the robot does the action itself.

I have explained that the normal reflex method is not suitable for speech-to-speech reflexes or proprioception-to-same-effector reflexes. Neither is the leading method appropriate for eye movements or speech effector movements. It is difficult to teach eyes to move by pushing them around. It is very difficult to push speech apparatus around so that it makes speech sounds.

The back-reflex method is not suitable for eye movements or arm movements because there are no corresponding stimuli for causing the movements. Neither (i) do visual stimuli correspond to eye movements, nor (ii) do proprioceptive limb stimuli correspond to limb movements, in the same way that speech stimuli and speech actions correspond. Leading is used when the "trigger" stimuli and their

"triggered" actions belong to the same effector. The stimuli that result from limb actions depend on the interaction of the limb with the physical environment. So limb motor commands and the resulting proprioceptive signals do not correspond as much as speech actions and stimuli do. Back-reflex is used for stimuli and actions which have the sort of one to one relationship that speech sound actions and stimuli have. Under all but the most extreme conditions, we hear just what we speak.

Arm movement reflexes that are triggered by stimuli other than arm movement stimuli, may be put into a robot. For example a robot could have a reflex to automatically withdraw its arm whenever near-damaging forces are sensed by touch or force sensors in its hand. However, it may be very difficult to provide a robot with arm reflexes which would enable it to do all the movements required for all tasks. While eyes do not have to compensate for different loads, arms do. Chapters III, IV and V show the process of leading a robot through task movements to be a possible way for the robot to acquire motor skills.

VII.3.2 Conditions for a reflex, back-reflex or led action

Leading a robot's arm causes actions to be remembered only if the robot's arm is relaxed. Similarly, reflexes might cause reflex actions to be performed only if the robot isn't doing other incompatible actions. Reflex eye movements might occur only if the robot isn't moving its eyes "voluntarily". Mimic-speech, or back-reflex speech, has been remembered in an MCLS only when no robot speech action preceded the speech stimulus (Andreae, 1977a).

VII.4 CONCLUSION

The leading, guiding and teacher-programming methods may not be suitable for teaching a robot some movements; eye movements for example. Instead, such actions might be preprogrammed to occur as reflex-actions, actions triggered by reflex-trigger stimuli.

Actions that are first performed by reflex alone can be learned by an MCLS and then performed when the reflex-trigger is not present. Reflexes in an MCLS can be generalized to a wide range of non-reflex situations. The MCLS PURR-PUSS does not overgeneralize the reflexes it learns.

A head movement system may be required on a robot with a moving eye. The leading method, with a resistive training mode, might be used for teaching head movements to a robot.

Reflexes are suitable for the learning of eye actions, but not for the learning of speech and arm actions. Leading is suitable for the learning of arm actions, but not for the learning of eye and speech actions. Back-reflex is suitable for the learning of speech actions, but not for the learning of eye and arm actions.

NOTES FOR CHAPTER VII

1. In fact, I want to provide the robot with reflexes that are the minimum preprogramming necessary for the learning of all actions to be possible. Superfluous preprogramming may interfere with learning. Therefore, given some reflexes, it must be possible for the robot to learn to use the resulting actions in many situations, including those where the reflex-triggering stimulus is not present.

2. The teacher may actually type "Hello" at the MCLS with a keyboard, rather than speaking the word "Hello". It makes no difference to the MCLS in so far as its stimuli are preprocessed sensory information, so the stimuli it could get as a result of hearing speech could also result from typed characters. Being able to use spoken words will make quite a difference to the interaction between the MCLS and the teacher. However, back-reflex works the same in either case.

3. In humans, "... the visual system must be able to steer the eyes so as to obtain a number of orderly foveal inspections". (p.553. Didday & Arbib, 1975).

4. PURR-PUSS was simulated on an EAI 640 computer. Except for the teacher, the environment I used for PURR-PUSS was simulated by computer programs. PURR-PUSS is open to the real world because the teacher is connected. PURR-PUSS can have real environments (see Rushby et al, 1975; Palfi, 1977a).

5. If two or more actions are given priority 6, and none are given a higher priority, then one of those actions will be selected randomly for performance. In fact this is what happens in the illustrative interaction, when the *B look-left action is performed by PURR-PUSS. Both *B and [R, a turn-right action, are given a priority of 6. Over a number of runs of the illustrative interaction, *B and [R will each be chosen about half of the time. This highlights the fact that the reflex-action *B is being performed by PURR-PUSS with the minimum amount of prediction. The *B is given the lowest priority, 6, and even then is performed only sometimes. Of course, once the reflex-action has been performed, more associations will be formed with it. These associations will enable the reflex-action to be performed with a higher priority than 6.

6. Dummy events have been used by Palfi (p.6 1977b).

7. The second-order system derives its name from the second-order differential equation that mathematically describes it (Di Stefano et al, 1967):

$$\ddot{x} + b\dot{x} + cx = f(t) \qquad (1)$$

where x is a variable, the head angle of a robot for example, $\dot{x}$ and $\ddot{x}$ are the first and second time derivatives of x, b and c are constants, and f(t) is a forcing function. (1) may be written

$$\ddot{x} + 2zw_o\dot{x} + w_o^2 x = f(t) \qquad (2)$$

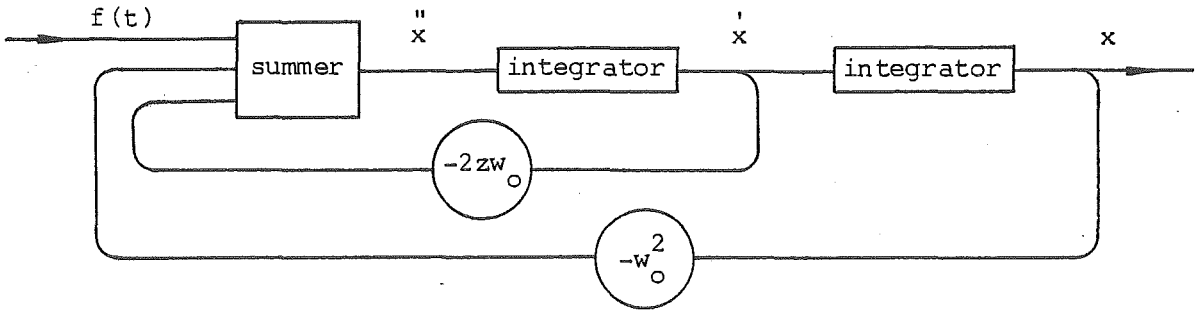
where w_o is the undamped natural frequency and is constant, z is the damping factor and is constant, and w_n is the natural frequency,

$$w_n = w_o \sqrt{1 - z^2} \; . \qquad (3)$$

(2) can be rewritten as

$$\ddot{x} = f(t) - 2zw_o\dot{x} - w_o^2 x, \tag{4}$$

and represented diagramatically as shown below.



APPENDIX FOR CHAPTER VII

Section A.1 explains how reflexes may be learned by PURR-PUSS. Section A.2 gives an illustrative interaction. Section A.3 discusses PURR-PUSS's learning of reflexes in general. Section A.4 describes the head movement system with a resisting training mode.

VII.A.1 Reflex learning in PURR-PUSS

Prior to the work described in this chapter, first reported in MacDonald (1979), PURR-PUSS was given its first memory of actions by the teacher forcing actions into it, and by mimic-speech (Andreae, 1977a). The teacher's forcing of actions is a guiding method, as described in section 3.2 of chapter II. Using this guiding method, the teacher, the person interacting with PURR-PUSS, could force an action on PURR-PUSS whenever PURR-PUSS did not select an action. In this way, PURR-PUSS was made to perform and remember new actions. I have already discussed, in chapters III, IV and V, the leading method, which enables a teacher to use his own natural abilites to teach arm actions to an MCLS. I pointed out in section 1.1 that guiding, leading and programming-by-the-teacher may not be suitable for teaching some movements, for example eye movements, to a robot.

Figure VII-10 shows the context templates for the rules used by PURR-PUSS in the SQUARES environment. All actions and stimuli of PURR-PUSS were combinations of DECwriter symbols.[4] The production templates of PURR-PUSS have homogenous events in them. That is (i) each context is a number of events of the same type, and (ii) each context predicts an event of the same type as the events in the context. For example, stimulus contexts predict stimuli. In some MCLSs, contexts of one type of event predict another type of event. For example the arm-robot MCLS of chapter V has in it contexts of stimuli which predict actions.

Context Type	Description of Events in Context	Length of Context
Timing Contexts		
Main	all action-stimulus pairs	3 pairs
Action	all actions	7 actions
Stimulus	all stimuli	7 stimuli
Threading Contexts		
SQRS [	[action-stimulus pairs	4 pairs
SQRS *	*action-stimulus pairs	3 pairs

Figure VII-10 Context Templates The context templates of PURR-PUSS, used for the interaction in Figures VII-15, VII-16 and VII-17 are shown. The main context receives all pairs, but [and * contexts receive only those pairs in which the first symbol in the action is [and * respectively. For example, [R/370 [L/070 [L/310 is the first main context in Figure VII-16. [R [L [L [F [F [R [F is the first action context. The first [context [R/370 [L/070 [L310 [F/300. 370 070 310 300 220 100 340 is the first stimulus context. *B/200 *C/40 *D/10 is the first * context.

Contexts in Agreement	Priority for action
Main, Action, Stimulus and [or *	4
Main, and any 2 of the others	6
[or *, Stimulus and Action	6

Figure VII-11 Decision Procedure

Priorities up to the top priority of 1 can occur using other contexts which are not used or important here.

PURR-PUSS's decision procedure is shown in Figure VII-11. In the decision procedure, priority 4 is a higher priority than priority 6.

The event sequence of Figure VII-9 could happen according to the priorities of Figure VII-11 if the reflex-action is predicted by at least three contexts. Then it would have a priority of 6 and would be performed if there were no other actions predicted with a priority of 6 or higher.[5]

Now, what contexts of PURR-PUSS could predict the reflex-action, as shown in Figure VII-9?

Just before the the reflex-action is performed in Figure VII-9 the contexts in the STM of the MCLS are:

Main context	<action>$_e$ <action>$_f$ <action>$_g$ <stim>$_e$ <stim>$_f$ <non-reflex-trigger>$_g$
Stimulus context	<stimulus>$_a$ ... <stimulus>$_f$<non-reflex-trigger>$_g$
Action context	<action>$_a$ ... <action>$_f$ <action>$_g$
[context	<[action> <[action> <[action> <[action> <[stim> <[stim> <[stim> <[stim>
* context	<* action> <* action> <* action> <* stim> <* stim> <* stim>

Three of these contexts must agree in predicting the <reflex-action>/<reflex-follower> pair for the reflex-action to be performed by decision of PURR-PUSS.

In sections A.1.1 through A.1.4 each context is examined in turn: main context (A.1.1); stimulus context (A.1.2); action context (A.1.3); and the two threading contexts, * and [(A.1.4).

VII.A.1.1 Main context The main context just before the reflex-action in Figure VII-9 is shown above. It cannot predict the reflex-action/reflex-follower pair. As shown in Figure VII-8 the main context that precedes a reflex has the reflex-trigger as the latest stimulus in the context. So productions stored that predict the reflex-action would have the reflex-trigger in the context. It was explained in section 1.3.2 that only productions stored in Figure VII-8, without the reflex-trigger in them, could predict the reflex-action in Figure VII-9.

VII.A.1.2 Stimulus context The stimulus context just before the reflex-action is performed in Figure VII-9, can predict the reflex-follower stimulus. As explained in section 1.3.2, a production to predict the reflex-follower in Figure VII-9 can be set up either (a) in the sequence of Figure VII-8, as long as the context of the production does not have the reflex-trigger in it, or (b) in another sequence, where neither the reflex-action nor the reflex-trigger occur. However, the stimulus production that is stored in Figure VII-8 to predict the reflex-follower, has the reflex-trigger in it. So only the second way, (b), can be used.

Also, for a stimulus production to predict the reflex-follower in Figure VII-9, it must have the non-reflex-trigger in it. Figure VII-12 shows the event sequence that must occur for the stimulus production to be set up. It must occur some time before the sequence of Figure VII-9, if the stimulus production is to predict the reflex-follower in Figure VII-9.

As shown by the sequence in Figure VII-12, a reflex-follower stimulus must be available following a non-reflex-action which itself must follow a non-reflex-trigger. When the sequence in Figure VII-12 occurs, a production is stored which has the required stimulus context given above, predicting the reflex-follower. Then the stimulus context will predict the reflex-follower in Figure VII-9.

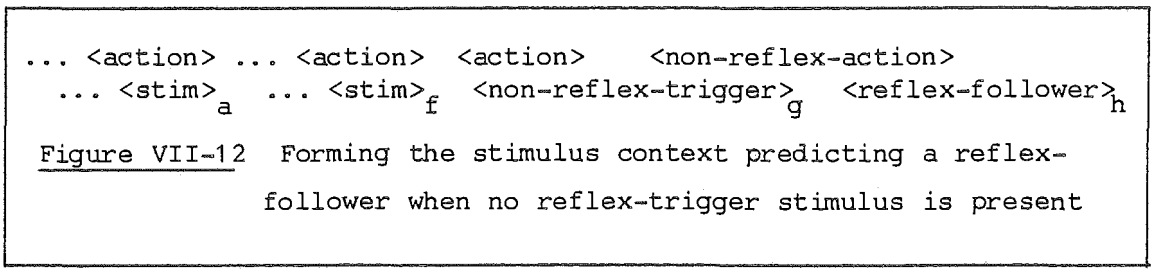


Figure VII-12 Forming the stimulus context predicting a reflex-follower when no reflex-trigger stimulus is present

[It is true that, once reflex actions have been performed by decision, "<non-reflex-action>" in Figure VII-12 can be replaced by "<action>". This is because then reflex-actions will be able to occur after non-reflex-trigger stimuli. However, the first reflex-action performed by decision must be performed without this replacement. The action before the reflex-follower in Figure VII-12 must not be a reflex-action.]

Finally in this section I shall discuss the sequence of events in Figure VII-12. The setting up of the stimulus production shown in Figure VII-12 has neither the reflex-action in it, nor the reflex-trigger. The non-reflex-action in Figure VII-12 may have no relation to the reflex-action in Figures VII-8 and VII-9, and yet the same reflex-follower stimulus is required to follow it. Otherwise that stimulus context could not predict the reflex-follower stimulus. Also, the Event diagram in Figure VII-12 may occur at any time before that in Figure VII-9 for the reflex-follower to be predicted in Figure VII-9 by the stimulus context. This occurrence of the reflex-follower stimulus, after a non-reflex-action that is unrelated to the reflex-action, will be referred to as an X-OCCURRENCE of the reflex-follower.

VII.A.1.3 Action context The action context just before the reflex-action in Figure VII-9 can predict the reflex-action. The reflex-trigger stimulus may be able to occur after <action>$_g$. For example, the environment might "interrupt" PURR-PUSS with the reflex-trigger, causing the reflex, regardless what action <action>$_g$ is.

VII.A.1.4 [and * contexts For a [threading context to predict the reflex-action in Figure VII-9 when the reflex-trigger is not present, the refex-action must be a [action and the reflex-trigger must not be a [stimulus. Otherwise a [context predicting the [reflex-action could not be formed in Figure VII-8 without the reflex-trigger being in it. [threading contexts predict only [actions. Similarly, for a * context to predict a reflex-action in Figure VII-9, the reflex-action must be a * action and the reflex-trigger must not be a * stimulus. * threading contexts predict only * actions.

[Note that in general MCLS productions do not have the same type of event in both the context and prediction of a production, as PURR-PUSS does.]

VII.A.1.5 Predicting a reflex-action

Summarizing A.1.1 through A.1.4, for the first performance of a reflex-action by decision as shown in Figure VII-9, only the stimulus context, action context and one of the threading contexts can predict the reflex-action.

Although the context templates shown in Figure VII-10 appear quite arbitrary, they were developed so that PURR-PUSS possessed particular properties, enabling her to learn important sorts of task (see Andreae, 1977a; 1977c; 1977d; 1978; 1979a; Andreae & Andreae, 1978; Andreae & Cleary, 1976). More recent MCLSs display a wider variety of context types, but there is no specific MCLS that has been developed and tested as much as PURR-PUSS (see Andreae, 1980a; 1980b; 1981; 1982a; MacDonald & Andreae, 1981; and also chapters V, VIII and IX).

I thus require agreement from the stimulus, action and either * or [threading contexts listed at the start of this section, A.1, for the performance of a reflex-action without its reflex-trigger stimulus. In the next section an illustrative interaction is given in which a reflex * action is predicted by action, stimulus and * contexts.

VII.A.2 An Illustrative Interaction

It was explained in the last section that an action, stimulus and * context can predict a reflex * action without the reflex-trigger being present, and so cause the reflex-action to be performed by decision. To confirm this, I adapted the simulated SQUARES environment, first used in 1974 (Andreae, 1974) and later used in the ROOMS task (Andreae, 1977a).[4] The SQUARES environment and the illustrative interaction are briefly explained below. A detailed explanation of the interaction is given in section A.2.1.

PURR-PUSS can do the actions [F, [R, [L and [G to move forward, turn right, turn left, and stay in the same place, respectively, in the SQUARES environment. The SQUARES environment is depicted in Figures VII-13, VII-14 and VII-15, and Figure VII-17 in section A.2.1. A shaded square is a wall. An unshaded square is an open space. D is a "dog". Arrows, ↑ or ↓ or ← or →, represent PURR-PUSS's current position and the direction she faces.

The four [actions are typed in by the teacher on a keyboard and are both sent to PURR-PUSS's "body", and remembered in productions. These actions could also have been led by the teacher, by the teacher pushing a mobile robot around, in a real environment.

Three other actions, *B, *C and *D, for look left, look ahead, and look right, respectively, are available to PURR-PUSS. However, these three "look" actions cannot be produced by the teacher typing them in. They are preprogrammed to occur by reflex, in the way shown in Figure VII-6. The three actions may be seen as corresponding to a robot moving its "eyeballs". They are unavailable to the teacher. As I pointed out at the start of this chapter, neither leading nor guiding would be suitable ways to teach a robot to move its eyes.

When a [F, [R, [L or [G is performed, the stimulus returned to PURR-PUSS is a three digit octal number, $X_1X_2X_3$, corresponding to the

binary bit pattern ABCDEFGH. F, G and H are 1 if a dog is to the left, ahead or right, respectively, and zero otherwise. A, B, C, D, and E are each 1 for a wall in the squares a, b, c, d and e, respectively, in Figure VII-13, and zero otherwise.

The actions *B, *C and *D return the previous SQUARES stimulus, the last $X_1X_2X_3$, masked by the octal numbers 200, 40 and 10, respectively. Thus, *B, *C and *D will result in 200, 40 and 10 if there is a wall to the immediate left, ahead or right, respectively, and 0 otherwise.

b	c	d
a	↑ PURR-PUSS	e

Figure VII-13
The squares which PURR-PUSS can see as walls or spaces in the SQUARES environment.

*B, *C and *D occur by reflex if X_3 in a [stimulus is 4, 2 or 1, respectively. That is, the *B, *C and *D occur if there is a dog to left, ahead or to the right, respectively.

An example of the SQUARES environment is shown in Figure VII-14.

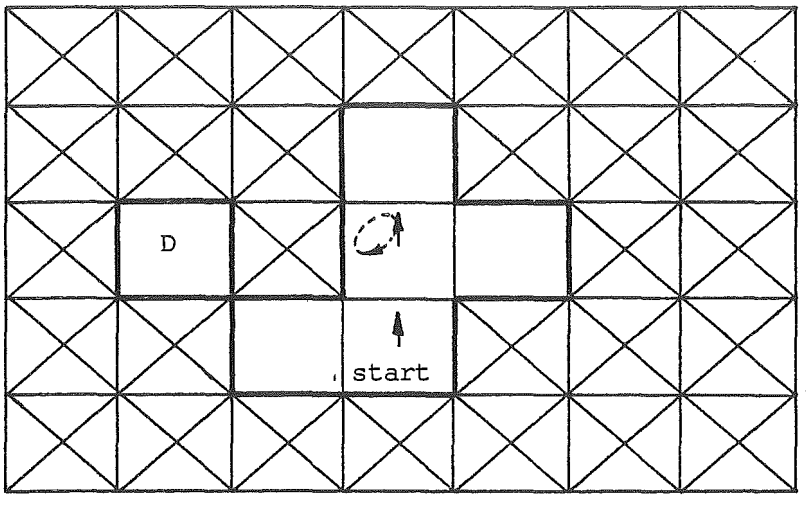


Figure VII-14 Example of the SQUARES environment

At the start, PURR-PUSS's last stimulus will have been 110. If she now does a [F action, then she will receive the stimulus 324, which will cause a *B by reflex. The stimulus 200 will be returned.

The dotted elliptical trace represents the *B "look" action.

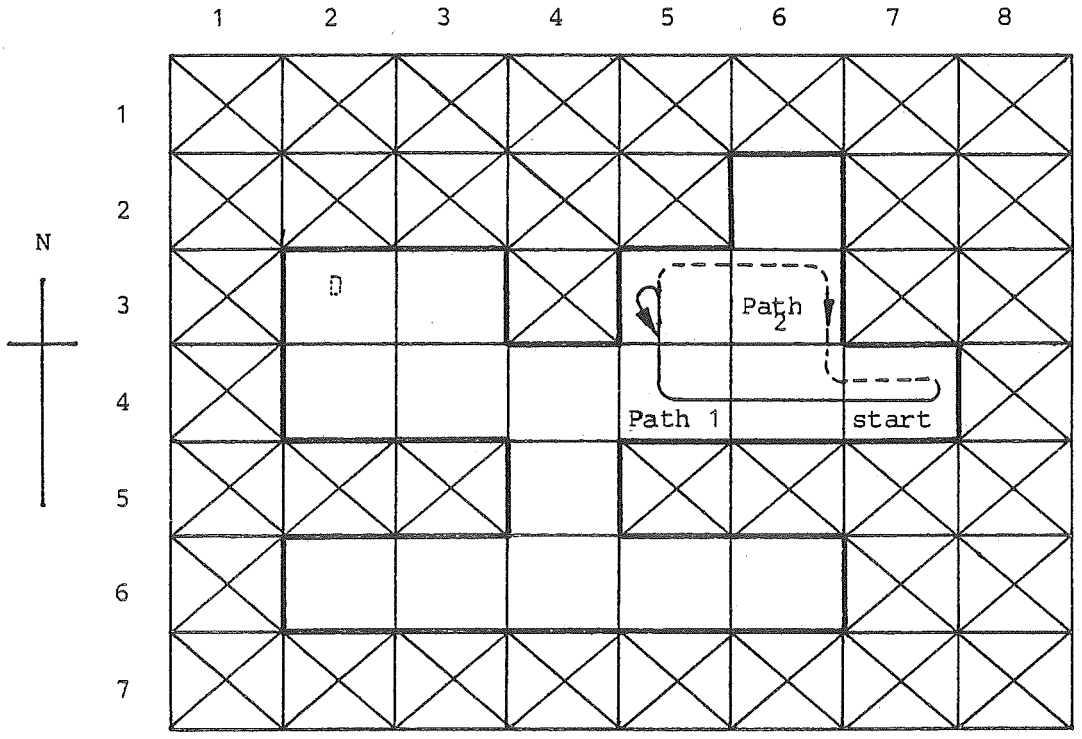


Figure VII-15 Summary of interaction with PURR-PUSS See the text for a brief explanation, and the appendix for a detailed explanation.

Figure VII-16 in section A.2.1 shows the interaction with PURR-PUSS where a *B is performed by decision on step 52. Figure VII-15 summarizes this interaction.

The interaction is explained in detail in section A.2.1. Briefly, from the "start" in Figure VII-15, PURR-PUSS goes along path 1 (steps 1-8 in Figures VII-16 and VII-17), then into path 2 (steps 9-15) and back to the start. This sets up a stimulus production which predicts the *B reflex-follower, 200. This production is association 1 in Figure VII-4 of section 1.

Then the dog is put into square (3,2). PURR-PUSS goes along path 1 (steps 16-22) and does a *B, because the dog is to her left, setting up an action production that predicts *B. This production is association 2 in Figure VII-4. PURR-PUSS then goes back along path 1 (steps 23-30) to the start, doing two more * reflexes to "look at" the dog, on the way.

PURR-PUSS goes along path 1 again (steps 31-37), doing *B again, which sets up a * production predicting *B/200. This production's context is the first three * reflex action/stimuli. It is association 3 in Figure VII-4.

Finally, the dog is taken away. PURR-PUSS goes along path 1. The stimulus, action and * contexts previously set up predict *B, and *B is performed by decision.[5]

VII.A.2.1 Interaction Figure VII-15 summarizes the illustrative interaction with PURR-PUSS, which was briefly discussed above. Figure VII-16 gives the actions and stimuli, and a detailed explanation of the interaction. Figure VII-17 gives a more detailed representation than Figure VII-15, of PURR-PUSS's movements in the SQUARES environment.

Summarizing the interaction in Figure VII-16,

(a) the 200 stimulus on step 10 in the stimulus context

370,070,310,300,220,100,340,

(b) the *B on step 22 in the action context [F,[L,[L,[F,[F,[R,[F, and

(c) the *B/200 on step 37/38 in the * context *B/200,*C/40,*D/10,

enable the agreement of stimulus, action and * contexts for a *B action on step 52, without having had the reflex-trigger stimulus 344 before it, on step 52.

VII.A.2.2 X-OCCURRENCE The stimulus context's prediction of 200 after the context 370,070,310,300,220,100,340 in Figures VII-16 and VII-17 is obtained by giving PURR-PUSS a [R on step 9. It is an X-OCCURRENCE of 200 that the [R causes 200, which is the stimulus given by *B on step 37. It is not always possible for the stimulus 200 to be obtained in this way. It depends on the environment. As shown in Figure VII-18, the last [stimulus of any stimulus context can be only

Step	Stim		Actn	Step	Stim		Actn	Step	Stim		Actn
1	-		-	19	300	tchr	[F	37	344	reflex	*B
2	-	tchr	[R	20	220	tchr	[R	38	200	P4	[L
3	370	tchr	[L	21	100	tchr	[F	39	072	reflex	*C
4	070	tchr	[L	22	344	reflex	*B	40	040	P4	[L
5	310	tchr	[F	23	200	tchr	[L	41	011	reflex	*D
6	300	tchr	[F	24	072	reflex	*C	42	010	P4	[F
7	220	tchr	[R	25	040	tchr	[L	43	140	P4	[L
8	100	tchr	[F	26	011	reflex	*D	44	030	P4	[F
9	340	tchr	[R	27	010	tchr	[F	45	130	P4	[F
10	200	tchr	[F	28	140	tchr	[L	46	370	P4	[L
11	140	tchr	[R	29	030	tchr	[F	47	070	P4	[L
12	200	tchr	[G	30	130	tchr	[F	48	310	P4	[F
13	200	tchr	[F	31	370	tchr	[L	49	300	P4	[F
14	160	tchr	[L	32	070	tchr	[L	50	220	P4	[R
15	130	tchr	[F	33	310	tchr	[F	51	100	P4	[F
16	370	tchr	[L	34	300	tchr	[F	52	340	P6	*B
17	070	tchr	[L	35	220	tchr	[R	53	200		
18	310	tchr	[F	36	100	P6	[F				

Figure VII-16 Interaction with PURR-PUSS

Most of the actions in the Figure are performed by the Teacher (tchr). Six of the actions are caused by reflex. The remaining actions are the result of PURR-PUSS's decisions and are marked with P4 or P6, according to which priority applied: P4 for for four agreeing contexts; P6 for three agreeing contexts.

PURR-PUSS starts, on step 1, facing North in square (4,7), as indicated in Figure VII-17 (a). Teacher does [R for PURR-PUSS on step 2. Seven stimuli, 370, ... ,340, and six actions, [L, ... ,[F, occur from step 3 to the action on step 8 and the stimulus on step 9. These are the same as the stimuli and actions from step 46 to the action on step 51 and the stimulus on step 52, as shown in Figure VII-17 (d). The stimulus 200, an X-OCCURRENCE, follows on step 10. So on step 52 the stimulus context predicts the stimulus 200 for step 53. However, the action context from step 2 to step 8, which is followed by [R on step 9, does not match the action context from step 45 to step 51. So action context does not predict [R for step 52. This is because step 2 has [R while step 45 has [F, even though they both give the same stimulus.

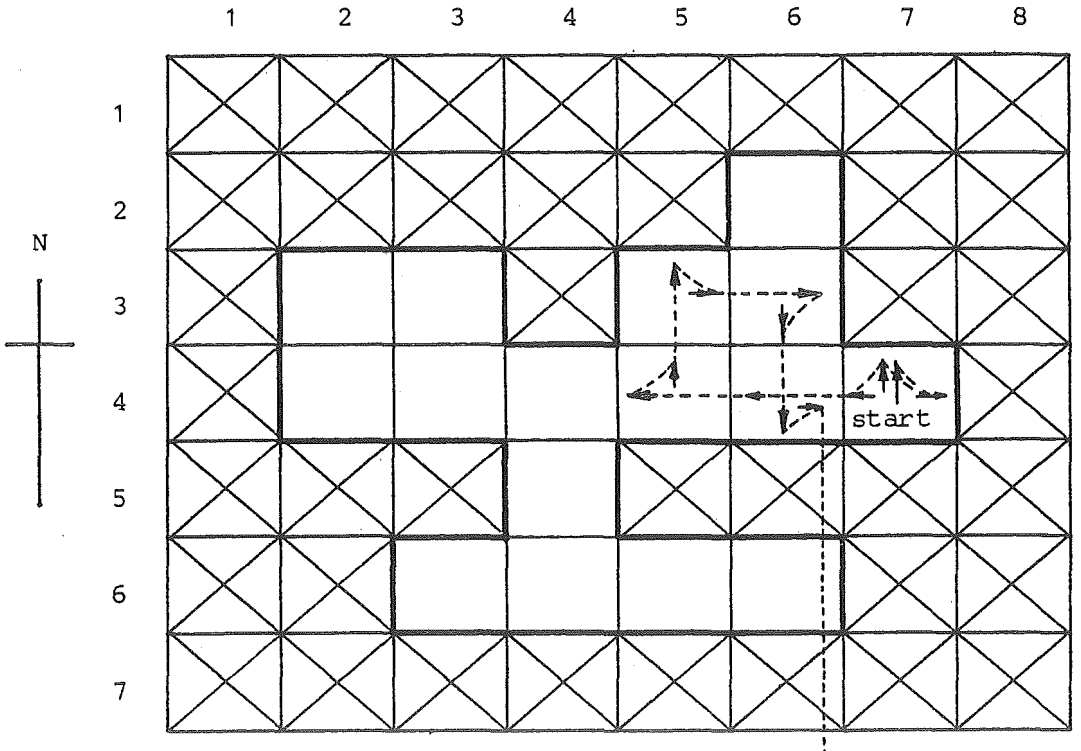
Steps 10 to 15 bring PURR-PUSS back to square (4,7). The action/stimulus sequence on steps 15-22 sets up the action context [F,[L,[L,[F,[F,[R,[F, predicting *B. Just before step 21, PURR-PUSS is put into "suspended animation" and the dog is put into square (3,2), as shown in Figure VII-17 (b). The dog "runs into the room" while PURR-PUSS is "not looking". This action sequence predicts *B on step 52 from the actions on steps 45-51.

After step 38, (a) the *B/200 on step 22/23, (b) the *C/040 on step 24/25, and (c) the *D/010 on step 26/27 make up the * context before the *B/200 on step 37/38. Therefore. after *D/010 on step 41/42, which is preceded by *B/200 on 37/38 and *C/040 on 39/40, the * context predicts *B/200. Just before the [F on step 51, PURR-PUSS in put into suspended animation again and the dog removed, since we hope that a *B will occur by decision and not reflex.

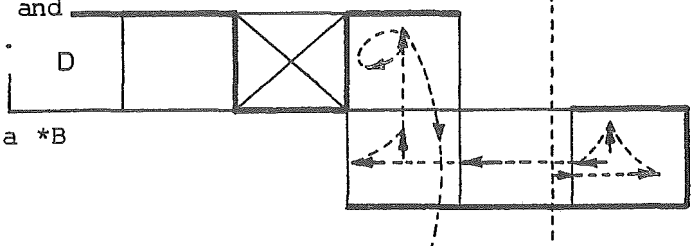
When the action context predicts *B and the stimulus context predicts 200, on step 52, they confirm the * context prediction for *B/200 as the next *action/stimulus. Hence there is enough agreement for a decision and *B is selected with priority P6 on step 52.

Figure VII-17 The movements of PURR-PUSS in the SQUARES environment for the interaction of Figure VII-16. Only the squares that PURR-PUSS moves in are shown in (b), (c) and (d).

(a) Steps 2 through 14 of the interaction in Figure VII-16.

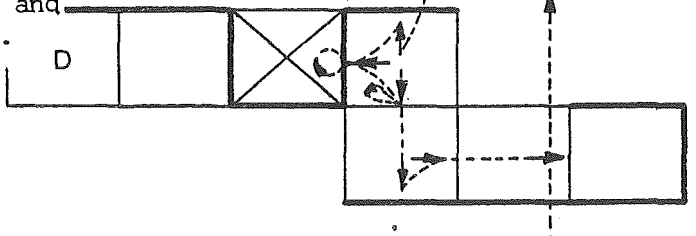


(b) Steps 15 to 22 and 30 to 37.



represents a *B

(c) Steps 23 to 29 and 38 to 44.



(d) Steps 45 to 52.

The DOG, D, is removed on Step 51

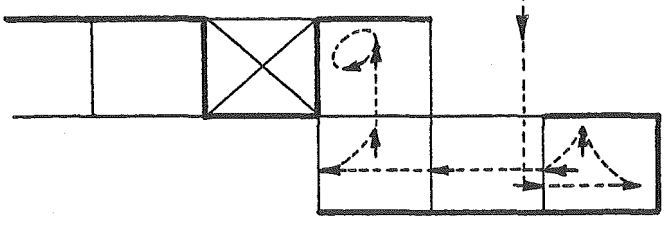


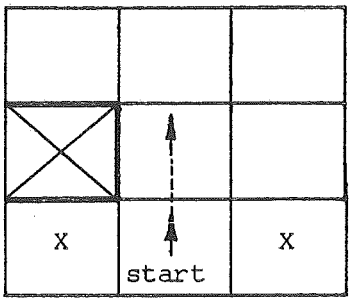
Figure VII-18 Possible [stimuli which precede [F, [R, [L or [G, to return the stimulus 200 or the stimulus 000.

The last digit is always zero since no reflex-action is wanted. The last [stimulus is restricted to being 14 out of 32 possibilities. An "X" in a square means that it may be wall or space, that is 1 or 0.

(a) Before [F/200

Possible stimuli preceding [F are 100, 110, 300 and 310.

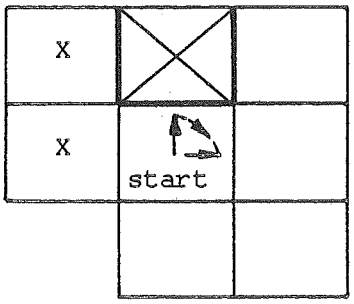
After [F the stimulus 200 is returned to PURR-PUSS.



(b) Before [R/200

Possible stimuli preceding [R are 040, 140, 240 and 340.

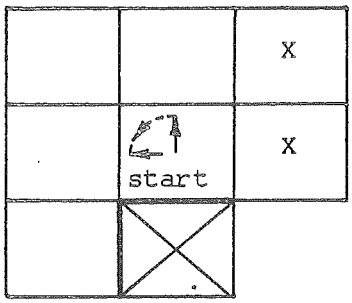
After [R the stimulus 200 is returned to PURR-PUSS



(c) Before [L/200

Possible stimuli preceding [L are 000, 010, 020 and 030.

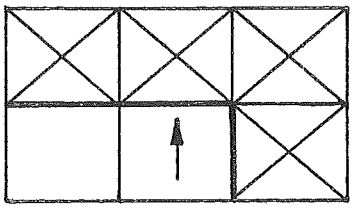
After [L the stimulus 200 is returned to PURR-PUSS.



(d) The stimuli that may precede [F, [R or [L if 000 is to follow are given by the Figures in (a), (b) and (c), but with the walled in square in each one clear. 000, 010, 200, 210 may precede [F. 000, 100, 200, 300 may precede [R. 000, 010, 020, 030 may precede [L. Only 200 and 210 are added to those in (a), (b) and (c).

(e) A [G/200, or [G/000, may be preceded only by 200, or 000, respectively.

(f) A situation where a *B is reasonable, but cannot be predicted by a stimulus context whose latest stimulus is a non-reflex, [stimulus
A [R, [L, [F or [G in this situation could return neither 200 nor 000.



(a) 100,110,300,310 with a [F following, (b) 040,140,240,340 with a [R following, or (c) 000,010,020,030 with a [L following, for the next stimulus to be 200. So not any stimulus context can be set up to predict 200 if there is a non-reflex, [action before the 200. *B can also be followed by the stimulus 000. If the stimulus context predicted this, then it could be set up to agree with the * context in a manner similar to that in Figure VII-16 for the stimulus 200. However, only the additional [stimuli 200 and 210, may immediately precede a [R, [L or [F which is followed by 000, as explained in Figure VII-18(d). Only the stimulus 200, or the stimulus 000, may precede a [G which is followed by 200, or 000, respectively.

In only 14 of 32 possible SQUARES environment situations can a [action be followed by either 000 or 200, the reflex-followers of *B.

VII.A.3 Discussion

VII.A.3.1 X-OCCURRENCE In the interaction in Figures VII-15, VII-16 and VII-17, in which *B is performed by decision, the stimulus production which predicts the *B-reflex-follower 200, was formed by X-OCCURRENCE. The 200 occurs after a [action, not the *B, just as required by section A.1.2, and depicted in Figure VII-12.

The other *B-reflex-follower is 000. It is not always possible to get 200 or 000 after a [action. As explained in section A.2.2, 200 or 000 may occur after a [action in only 14 of 32 possible SQUARES environment situations. So the *B could not be performed by decision in 18 SQUARES environment situations, if the 200 or 000 must immediately follow another visual stimulus and a [action.

However, the stimuli in the stimulus context do not have to be [stimuli, as they are in section A.2. The action before the reflex-follower does not have to be a [action. For predicting a stimulus 200, without the reflex-trigger present, we can have the

stimulus context

<non-[stimulus>$_1$ ... <non-[stimulus>$_q$ <non-[-non-reflex-trigger>$_r$.

Then the sequence of events for the X-OCCURRENCE of the stimulus 200 is the one shown in Figure VII-19, rather than the more general one shown in Figure VII-12. The most recent [action/stimulus event can be arbitrarily far back in the event sequence, as long as it is more than seven events earlier, to be out of the stimulus context. In this way the stimulus and action contexts do not depend on the latest [action/stimulus event when they predict a reflex-action/reflex-follower pair. Once (a) the action context of Figure VII-19 (<non-[action> ... <non-[action>) predicts *B, (b) the stimulus context of Figure VII-19 (shown above in this paragraph) predicts 200, and (c) a * context predicts *B/200, the *B can be performed independently of the most recent [action/stimulus. Therefore the *B can be performed in <u>any</u> situation in the SQUARES environment.

<latest [action> .. <non-[action> .. <non-[action> <non-reflex-action>
<latest [stim> .. <non-[stim>$_1$ .. <non-[-non-reflex-trigger>$_r$ 200

<u>Figure VII-19</u> Formation of a stimulus production with non-[stimuli predicting 200. Compare this Figure with Figure VII-12.

Furthermore, it is shown in section A.3.2 that the restricted "latest" [stimulus, shown in Figure VII-19, need precede the reflex-action by only k non-[stimuli <u>once the reflex-action has been learned</u>. k is the length of the main context, which is 3 in PURR-PUSS.

<u>VII.A.3.2 General learning of reflexes</u> Sections A.1 and A.2 showed (a) that one way for actions to be first performed and remembered by the MCLS PURR-PUSS is by reflex, and (b) that this can lead to the performance of the reflex-action without the reflex-trigger's presence. However, for this to happen, the arrangement

of the reflex mechanism within the multiple context of PURR-PUSS is severely restricted:

(a) Threading contexts. The reflex-trigger stimulus and the reflex-action must be in different types of threading context, so that the context which predicts the reflex-action can do so without having the reflex-trigger in it (see section A.1.4);

(b) Action context. The environment must be able to interrupt PURR-PUSS to give a reflex-action after any action which we want to be followed by the reflex-action. Then an action context predicting the reflex-action can be set up. (see section A.1.3);

(c) Stimulus context. Any stimulus that follows the reflex-action must also be a stimulus that can follow an action not caused by reflex, so that a stimulus context can predict the reflex-follower stimulus without having a reflex-trigger stimulus as its newest event. In the interaction of Figure VII-16 both reflex * actions and non-reflex [actions are followed by the same sort of stimuli. Some stimuli that occur after [actions can occur after * actions. (see section A.1.2);

(d) it follows from (c) that at least one non-reflex-action must be available, so that the first reflex-action can be performed by decision. Therefore, all of PURR-PUSS's actions cannot begin as reflexes. Also, one reflex-follower stimulus per reflex-action must be available after at least one action which is not performed by reflex. That is, at least one X-OCCURRENCE of at least one reflex-follower stimulus must be possible for each reflex-action that is to be learned.

Now that I have established that reflex actions can be learned and performed by decision, without being triggered by trigger stimuli, I wish to explain how general the learning of reflex actions is. Can the reflex-action *B be performed by decision, as it is in Figure VII-16,

in every reasonable situation? That is, can it be performed in those contexts or situations where one would expect it?

Section A.3.1 explained that the reflex-action *B can be learned in every SQUARES environment situation that can occur, but seven non-[events must precede the reflex-action in 18 of the 32 possible situations.

To discuss this in more general terms, let us consider contexts with general, but fixed, lengths. Then the requirement for seven intercalating stimuli becomes a requirement for k_s intercalating stimuli. k_s is the length of the stimulus context. Let the length of the main context be k.

Suppose $k < k_s$, as it is in PURR-PUSS. Once the reflex-action has occurred without the reflex-trigger being present, there will be a main context that predicts the reflex-action, but which does not have the reflex-trigger in it. This main context is given in section A.1 for k = 3, and is the main context just before the reflex-action is performed in Figure VII-9. Its k stimuli will be the same as the latest k stimuli in the stimulus context that predicted the reflex-follower. Thus, that main context's prediction of a reflex-action/reflex-follower pair will be of the form

$$\begin{array}{l} \langle act\rangle_1 \ \ldots \ \langle act\rangle_{k-1} \ \langle act\rangle_k \\ \quad \langle stim\rangle_1 \ \ldots \ \langle stim\rangle_{k-1} \ \langle non\text{-}refl\text{-}trig\rangle_k \end{array} \quad \text{predicts} \quad \begin{array}{l} \langle refl\text{-}act\rangle \\ \quad \langle refl\text{-}foll\rangle \end{array}$$

For a * reflex-action to be performed by decision with a priority of 6, either

(a) stimulus, action and *,

or (b) stimulus, main and *,

or (c) stimulus, main and action,

or (d) main, action and *

contexts must be predicting the * reflex-action, as shown in

Figure VII-11. In each case, a prediction is needed from a main context, a stimulus context, or both. Both contexts include the last k stimuli to occur before the reflex-action.

Consider the k stimuli in a main context that predicts a reflex-action/reflex-follower pair. These k stimuli must have been the latest k stimuli in a stimulus context that was immediately followed by the X-OCCURRENCE of the reflex-follower. This is for two reasons.

Firstly, there is only one way for the main rule to store a new production which predicts the reflex-action/reflex-follower pair, but which doesn't have the reflex-trigger in the context. Such a production can be stored in main rule only as a result of stimulus, action and * contexts predicting the reflex-action/reflex-follower, and so causing the reflex-action to be performed. This is case (a) above.

Secondly, the k most recent stimuli that are in the context of that new main production are the k most recent stimuli in the stimulus context which predicted the reflex-follower. That stimulus production which predicts the reflex-follower could have been set up in only two ways: (i) by the X-OCCURRENCE of the reflex-follower after that stimulus context, or (ii) by main, action and * contexts causing the reflex-action to be performed (case (d) above). In (ii) the main context's k stimuli must have come from a stimulus context in the first place, as just explained in the previous paragraph.

So the only sequences of k stimuli that can predict a reflex-follower stimulus, are those which can be followed by X-OCCURRENCE of the reflex-follower stimulus.

However, the $k+1^{th}$ to k_s^{th} last stimuli in a stimulus context are not restricted in this way. In case (d) above, these stimuli may not be in any of the agreeing contexts. They aren't in the action or main context, and may not be in the * context. The reflex-action may be performed, regardless of what the $k+1^{th}$ to k_s^{th} latest stimuli are. A

new stimulus production may be formed. The context of the stimulus production would have in it (i) k stimuli that had previously been followed by a reflex-follower, and (ii) $k_s - k$ stimuli that had not occurred before the k-stimuli-plus-reflex-follower. This production could agree in predicting the reflex-action/reflex-follower later on in case (a), (b) or (c) above.

Therefore, the restriction caused by the requirement for X-OCCURRENCE of reflex-followers means that after some stimuli, just those for which X-OCCURRENCE may occur immediately, the reflex-action can be performed by decision immediately. After other stimuli there must be at least k action/stimulus events, before a reflex-action is performed by decision. These k action/stimulus events could be "dummy" events, events that don't change the situation.[6] By "situation" I mean the event or events after which one wishes to get the reflex-action performed.

Thus the reflex-action can be learned quite <u>generally</u>, the k preceding action/stimulus pairs being "dummy" events, and <u>any</u> situation being represented by the $k+1^{th}$ to k_s^{th} earlier action/stimulus pairs.

Note that, if the main context is not shorter than the stimulus context, that is if $k \not< k_s$, then the number of dummies cannot be reduced to less than k_s.

In the SQUARES environment, using the contexts shown in Figure VII-11, an *B could be learned immediately after 14 of 32 [stimuli. After the other 18 [stimuli there must be <u>three</u> non-[action/stimulus events before the *B could be performed by decision. Main context's length, k, is three. The dummy events here must be non-[events or they alter the situation. They would change the most recent [stimulus, which one wants to be followed by *B.

It should be remembered that the requirement for X-OCCURRENCE is mainly a requirement on the robot's environment. For this and other

reasons, one must see the robot as a part of the wider system including its environment. Other reasons for the importance of this view of a robot are given in chapter VIII, in sections 2.2 and 7.

Finally, a difficult problem is to show that both (a) with reflexes PURR-PUSS is reasonably easy to teach, and (b) with reflexes PURR-PUSS learns reasonably easily. In particular,

1. one should not have to work out the contexts, events and predictions required for reflexes to be learned while one is interacting with PURR-PUSS. This would be tedious and unnatural.

2. There should be several ways to teach a task (Andreae & Andreae, 1978).

VII.A.4 Head Movement System and Results

In section A.4.1 the head movement system, shown in Figure VII-20, is explained. Section A.4.2 discusses some experimental results for the head movement systems's feasibility, stability and transient response characteristics.

VII.A.4.1 Head Movement System

In Figure VII-20, the head movement system is shown realized in three ways:

(a) servomotor control

(b) biological analogue

and (c) analogue computer simulation.

The system shown in Figure VII-20 (a) has an electric servo motor, which is approximately a second-order system when it is under closed-loop control[7] (Pipes & Harvill, 1970). In Figure VII-20 a "leaky" averager feeds back an average value of the position of the head. More recent positions are weighted more heavily during averaging; it is an exponentially weighted past average. Thus, if the position of the head is changed by physically moving the head and holding it for long enough, the system will slowly 'give up' fighting against the

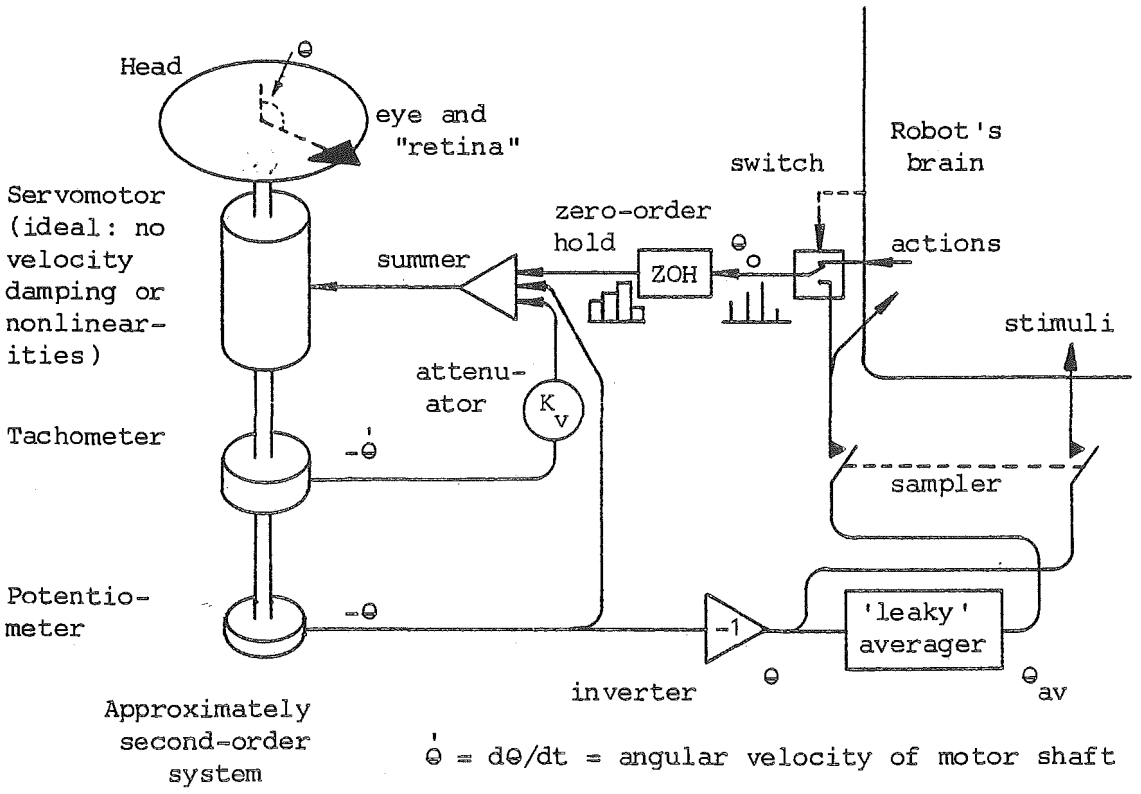


Figure VII-20 Head Movement System (a) Electrical Hardware

A servomotor controls the head angle Θ. Position and velocity feedback are employed. Also, an averager feeds back the average value Θ_{av} of Θ to the input of the "brain" via a sampler. These Θ_{av}'s would be remembered as actions, Θ_o, in the "head control" threading context of the robot "brain", in a similar way to the arm actions of the arm-robot of chapter V. Θs are the stimuli in the "head control" context. The Θ_{av} loop, which is positive feedback, is interrupted when a head control action is to be performed by decision. In this case the input to the zero-order hold is this "decision" Θ_o value, not the present Θ_{av} value. Following a "head control" action by decision, the input to the zero-order hold is, that action, until either the averager has had time to settle to the new value of Θ, in which case $\Theta_{av} = \Theta_o = \Theta$, or another head action is performed by decision. The ZOH converts a sequence of sampled signals into a continuous signal comprising a sequence of steps.

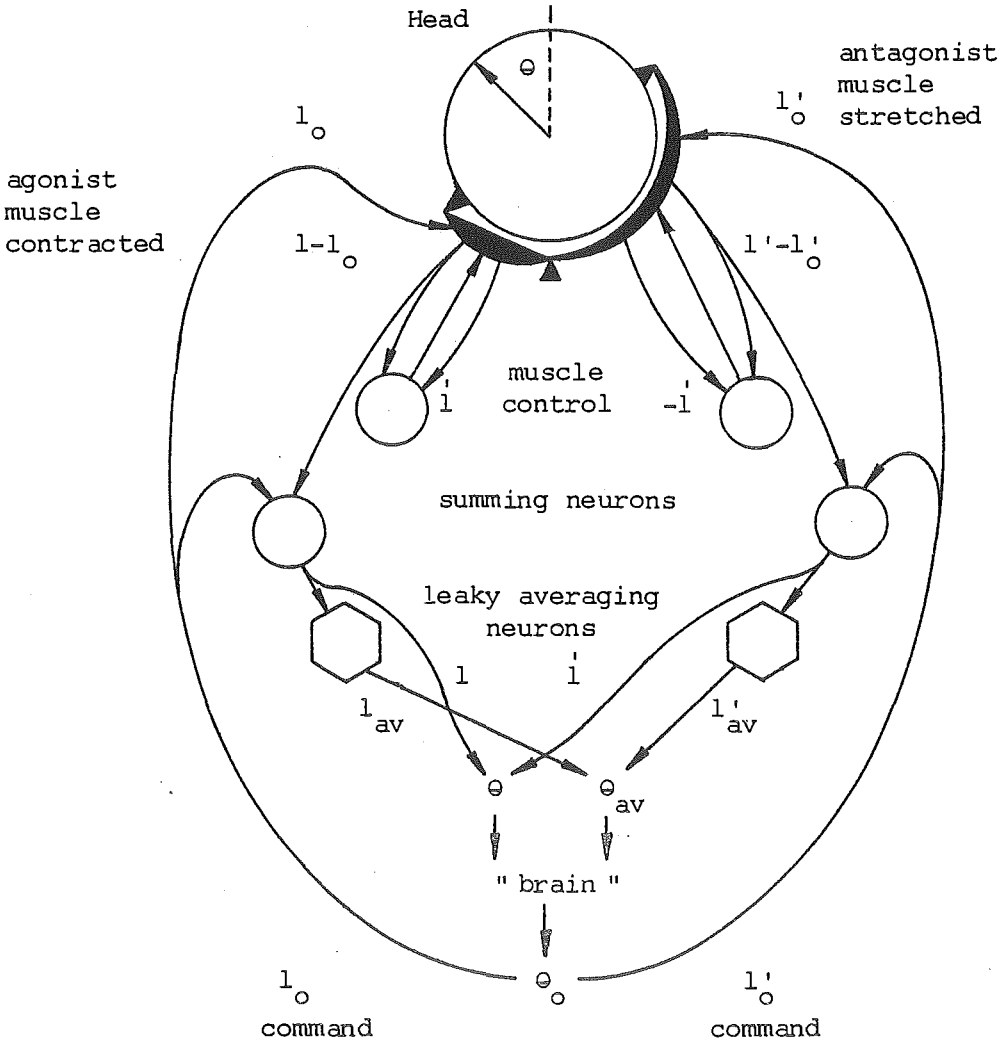


Figure VII-20 (b) Biological analogue of (a). A simple two muscle system controls the head. The muscles act only in one direction; they contract. The muscles respond linearly to excitation. Note that both the muscle length l, of the agonist, and the muscle length l', of the antagonist, are proportional to the head angle θ, since the muscles are stretched around the head. The spindle receptors inside the muscles are not shown in the Figure. They feedback velocity information, $\pm l'$, and position information, $l - l_o$ and $l' - l'_o$, with respect to the spindle rest lengths, l_o and l'_o. The spindle rest lengths are themselves set either by the averaged position or by decision actions from the "brain".

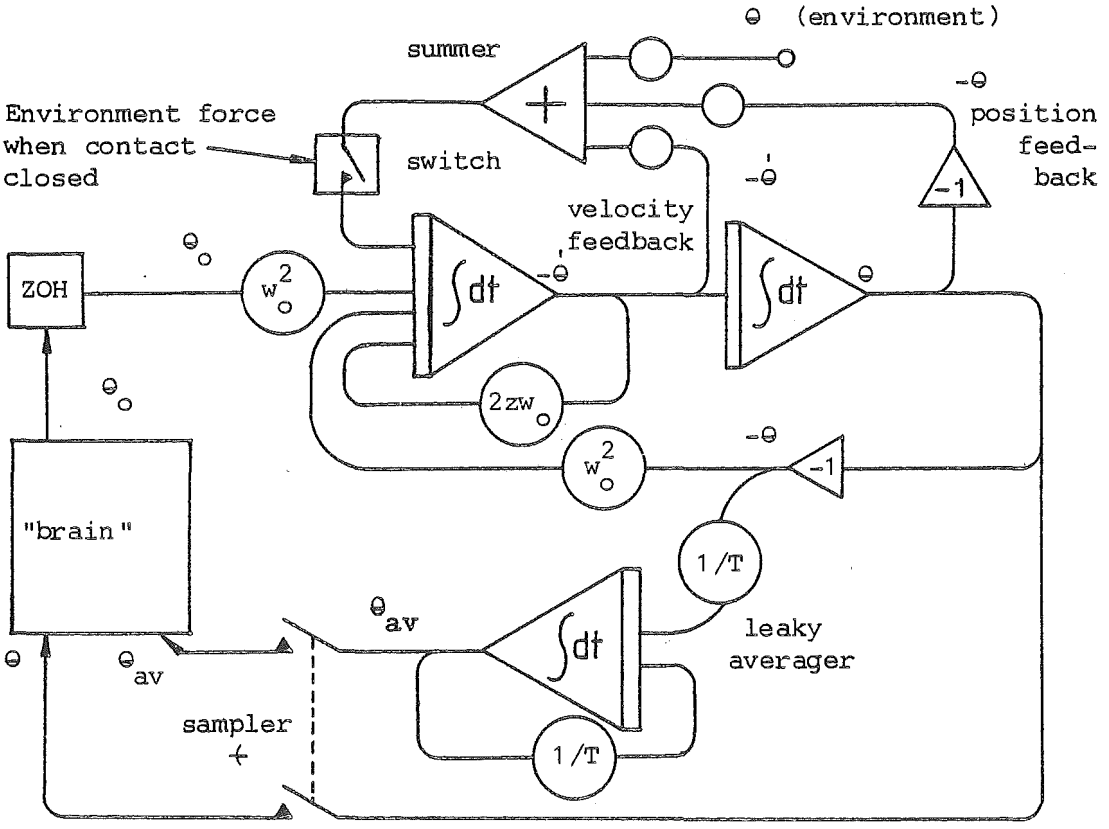


Figure VII-20 (c) Analogue computer diagram approximation to (a)

The environment force depends on:-

1. whether the environment is in contact with the robot's hand
2. the difference between the actual angle the head is at and the angle the environment "wishes" to set it at.
3. the velocity with which the head is approaching the θ (environment)

KEY[7]

z - damping factor of second-order system

w_o - undamped (z = 0) natural frequency of the system

$\dagger$ - the sampling period

T - the time constant of the low-pass filter (the leaky integrator).

w_n - ($= w_o \sqrt{1 - z^2}$) is the natural frequency of the system

$w_o >> 1/\dagger >> 1/T$

external force and allow the head to rest where it is being held. The slowly rising positive feedback from the leaky averager gradually cancels out the negative position feedback which causes resistance to the externally enforced position change.

The robot's learning system may interfere. A "head control" context, a threading context like * or [context, would receive stimuli and predict actions. The stimuli might be the angular head positions, Θ. The actions might be desired angular head positions, Θ_o. When an action is performed by the learning system, the switch in Figure VII-20 (a) will switch off the averaged positive feedback and switch in a desired head angle command, Θ_o. If no head movement action is decided on by the robot then the switch shown in Figure VII-20 (a) will switch Θ_{av}, the averaged Θ, into the control system input. If the environment causes Θ_{av} to differ from Θ_o by more than some threshold level, then the next action remembered is Θ_{av}. The corresponding stimulus is Θ. That is, if Θ_{av} varies by more than some threshold from Θ_o, then the robot's learning system can be 'interrupted' and given the action-stimulus pair, Θ_{av} / Θ.

This is a form of the leading method, which is discussed in chapters III, IV and V. The difference between this form of leading and that in chapters III, IV and V is that in the training mode the head is not relaxed. It still resists external forces.

Since an MCLS deals with time discretely, stimuli are sampled, and actions are sent to a zero-order hold (ZOH), as shown in Figure VII-20 (a). The time constant of the leaky averager is quite long. The control system will resist any force that is externally applied to the head for several seconds. It will slowly give up over a few "time units". A time unit is one sample period, $\dagger$. This behaviour is described in more detail below.

It might be necessary to disallow (a) the relaxing switch from switching to the averaged feedback signal, and (b) actions being learned from Θ_{av}, for a number of time units immediately following the performance of an action. This would give the averager time to catch up with the action performed. Alternatively, Θ_{av} could be set to Θ one time interval after an action is performed by decision. Otherwise the decision actions would be resisted by the averaged feedback until it had caught up from the old Θ_o to the new Θ_o. In fact the robot would learn to perform actions opposing the action it had just performed by decision, if the averaged feedback was not prevented from resisting learning system actions. This is because Θ_{av}, not having caught up to the new Θ_o, would be switched to the control system in the time unit following the new Θ_o and would bring Θ_o part way back to the old Θ_o!.

VII.A.4.2 Experimental Results In Figure VII-21 is shown a control system equivalent to the system in Figure VII-20, but without a robot "brain". This system was investigated in the laboratory. The motor shaft may be rotated by hand and held in a new position. This results in an error signal at the summer output; hence the motor opposes the external force. Slowly the Θ_{av} feedback loop is driven towards Θ by the leaky integrator. Thus the error, $\Theta_{av} - \Theta$, is driven towards zero. If the shaft is held until Θ_{av} reaches Θ, then released, it remains in that position Θ. A trace of this behaviour is shown in Figure VII-22 (a). Should the shaft be released before Θ_{av} reaches Θ then the second-order system drives Θ to Θ_{av}, since the response time of the motor system is much faster that that of the averager. Velocity feedback smooths the resulting changes in Θ. A trace of this behaviour is shown in Figure VII-22 (b). A trace of the response of the second-order system with inherent velocity feedback, K_{inh}, alone is shown in Figure VII-22 (c). With Θ equal to Θ_{av} the system is static.

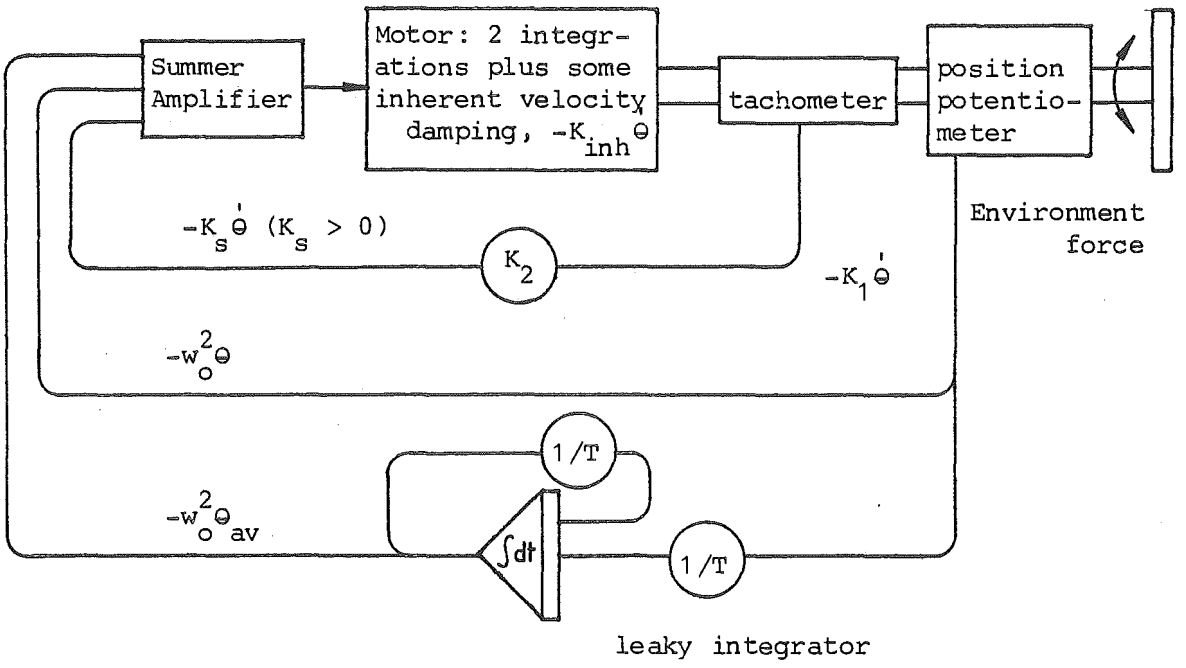


Figure VII-21 System investigated in the labororatory

This is approximately a second-order system[7] where

$$K_{synthetic} = K_s = K_2K_1,$$

$$K_v = K_s + K_{inh}$$

$$K_v = 2zw_o \quad \text{in Figure VII-20}$$

The system was investigated

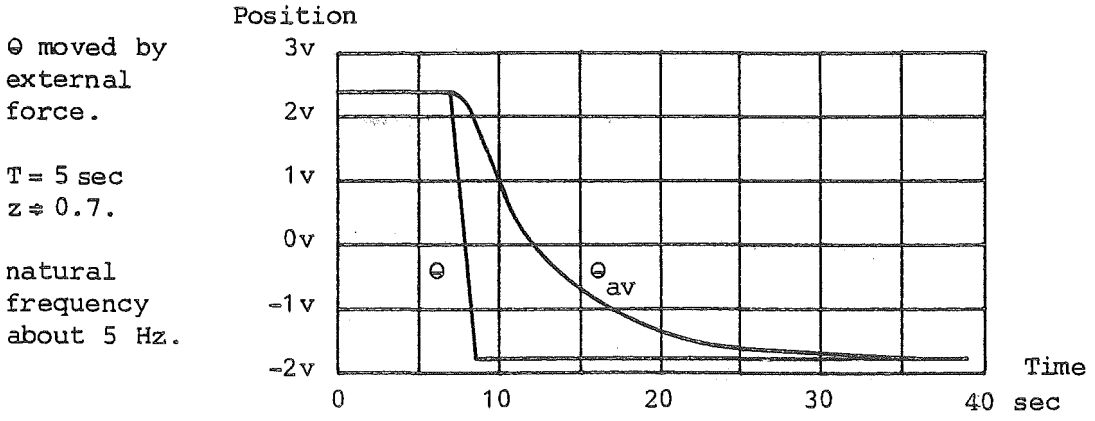
(a) for damping ratios from 0.1 to 2 (in Figure VII-20 the damping ratio is approximately 0.7)

(b) for T from 1 to 10 seconds,

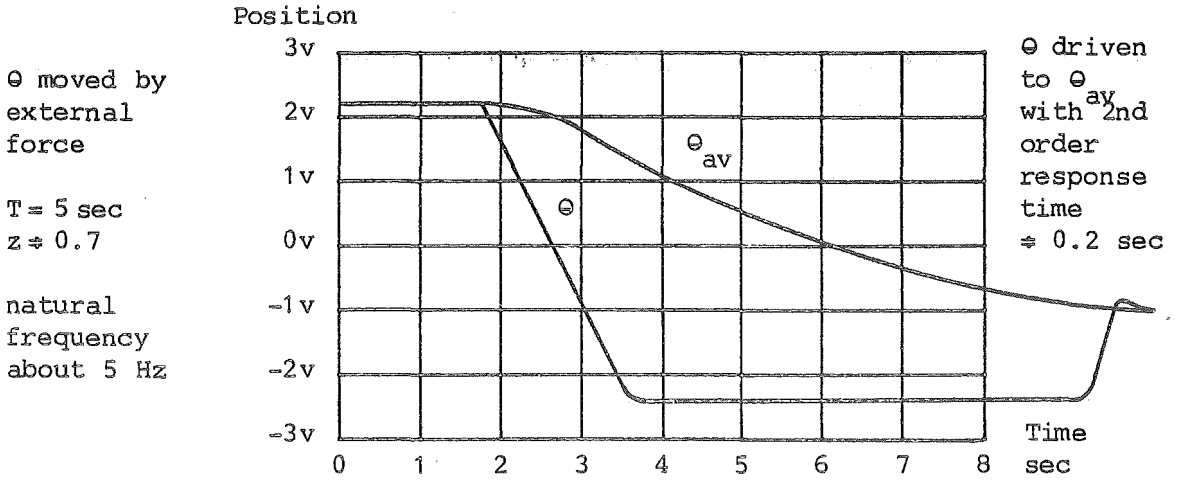
and (c) for w_o approximately 7 radians/sec.

Some results are shown in Figure VII-22.

(a) The Head is moved and held in the new position



(b) The Head is moved, held and then let go



(c) Step response of the second order system comprising the system of Figure VII-21, but with only inherent velocity damping, and no leaky integrator feedback.

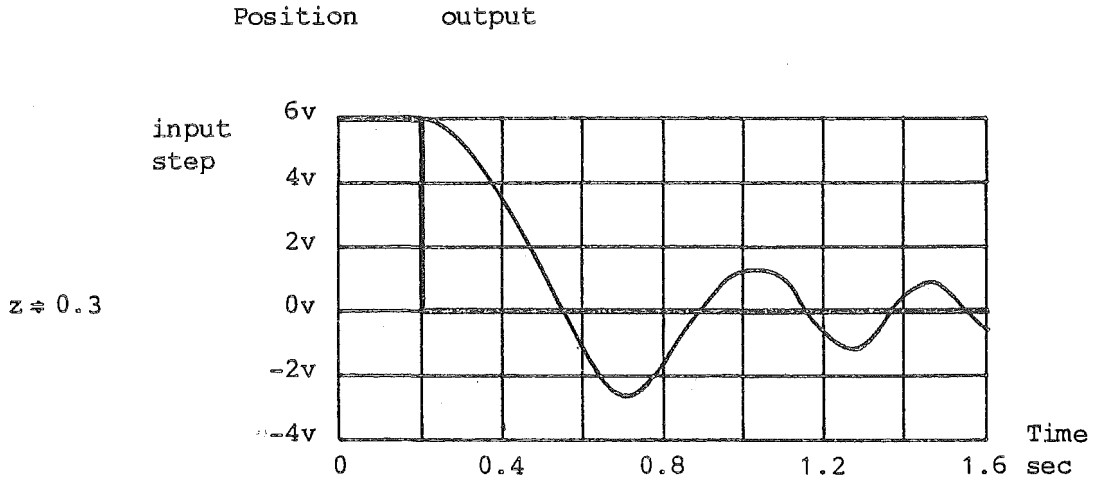


Figure VII-22 Traces of the behaviour of the head control system shown in Figure VII-21.

While static, the system was observed for ten minutes. During this time there was no drift in Θ. The system was stable under all conditions that the motor control system alone, that is without the leaky integrated positive position feedback, was stable. That is the system is stable as long as $z > 0$. z is the damping factor.

To use this system with an MCLS, two samplers and a ZOH would be required, as shown in Figure VII-20. If the sampling rate, $1/\dagger$, of those three components is much greater than 1/T, where T is the time constant of the leaky averager, then the output of the ZOH, which is the input to the input summer, will approximate the continuous Θ_{av} function. This is because Θ_{av} cannot change very much over the time $\dagger$. If the sampling rate is much <u>slower</u> than w_n, the natural frequency of the motor control system alone, then Θ <u>always</u> settles down to a steady value before the end of any sampling period.

Note that when a head movement action is performed by decision the Θ_{av} loop has been broken.

CHAPTER VIII

ANSWERING THE CRITICS

Hayes (1977) criticised multiple context learning systems (MCLSs), saying that the MCLS PURR-PUSS "... can only learn finite automata ..." and "So it certainly couldn't learn a grammar for any reasonably interesting subset of English ..." (p.10).

But grammars are finite sets of rules. So there is a finite automaton for every grammar (Miller & Chomsky, 1963).

An MCLS can learn any finite automaton that will fit into its memory (Andreae & Cleary, 1976; Andreae, 1977a; Andreae, 1977b).

So an MCLS can learn any grammar that will fit into its memory.[1]

Since the importance of an MCLS's being able to learn any finite automaton does not seem to have been recognized by the critics, as exemplified by the passage quoted from Hayes (1977) above, I explicitly show in this chapter that an MCLS can learn any grammar that will fit into its memory.

To do this I describe an MCLS that learns to simulate a simple automaton with a 'tape' memory. The tape memory is inside the MCLS. The tape can have a description of a grammar on part of it. The MCLS learns this description. The simulated automaton reads the description and implements the grammar. Another part of the tape is used as working memory, for intermediate results.

The automaton is what is called a "universal Turing machine" (UTM. Turing, 1936). Turing argued that a UTM, with an unbounded tape, could realize any process that could naturally be called an effective procedure. This cannot be proved, but it has stood the test of time and assault (Webb, 1980; Minsky, 1967; Kain, 1972; Hopcroft & Ullman, 1979).

A grammar, as a precise specification of the sentences in a language (Chomsky, 1963), is an effective procedure. So the MCLS can learn any grammar which fits on its finite "tape".

The finite memory of an MCLS does not limit it any more than humans are limited by their finite memories. There are two ways in which the finite memory of an MCLS can be seen as not being a limitation:

(a) The UTM-simulating MCLS (UTM-MCLS) can use a grammar on a sentence, as long as the tape is large enough to hold the grammar and provides enough memory to apply the grammar to the sentence. An MCLS can have enough memory for 70 years of learning and using a grammar, so the finite limit on the size of the tape should not prevent the MCLS from using any grammar that humans can use on any sentence that humans can use the grammar on. I expect a robot with an MCLS in it to have similar memory capacity to humans if it (i) has a lifetime expectancy of about 70 years, and (ii) can acquire and exercise human faculties such as the ability to communicate in real time with natural language;

and (b) It has been shown that an MCLS can use the environment as the tape for a UTM (Andreae & Cleary, 1976). So an MCLS can have an unbounded external tape available to it, as well as a finite internal one. It could learn and use any grammar, of any length, while the external tape is available to it.

Therefore I am justified in using MCLSs to implement reflexes and leading, in the face of Hayes's, and similar, criticisms.[2]

I am not suggesting that being able to learn every grammar that will fit into its memory, is all there is to an MCLS's learning to communicate with language. I am not suggesting that an MCLS needs to be able to learn every grammar that will fit into its memory, nor that an MCLS needs to be able to learn any particular grammar.

In section 1 grammars are briefly explained. In section 2 UTMs are explained. In section 3 the UTM-MCLS, which learns to simulate a UTM and some tape memory, is described. In section 4 the problems the UTM-MCLS has simulating the tape are discussed. In section 5 it is explained that the UTM-MCLS of section 2 could have been taught using reflexes or leading. In section 6 some brief comments about the syntactical characteristics of the UTM-MCLS are made. In section 7 the question "What does an MCLS's being able to learn any grammar show?" is addressed. In section 8 the question "Does an MCLS need to be able to learn every grammar?" is addressed.

Andreae and MacDonald (1981) also state the importance of the UTM's simulation by an MCLS. Andreae (1980a), and MacDonald and Andreae (1981) discuss another MCLS which can simulate a UTM and some tape. It is a parallel MCLS, derived by John Andreae from the one discussed in section 4, which itself was first described in MacDonald (1980). Parallel MCLS's do actions in parallel sequences, instead of performing just a single sequence of actions. The parallel MCLS simplifies my demonstration somewhat. I will mention some of these simplifications. In particular, my UTM-MCLS might not be able to learn all the internal memory, or "tape", needed even for a finite lifetime. The parallel MCLS can, since it learns to count tape in modulo-N. Also, I point out that (a) a TM starts with its tape ready made, while both MCLSs have to build their own tapes, and (b) if the length of the tape that the MCLS learns is a linear function of the input sequence length, then the MCLS can learn and implement a "linear bounded automaton", which can realize more grammars than a finite automaton can realize, but fewer grammars than a UTM can realize (Kain, 1972).

VIII.1 GRAMMARS

Grammars are briefly explained in this section. For more detailed descriptions, the reader is referred both to standard texts on the subject, for example, Minsky (1967), Kain (1972) and Hopcroft and Ullman (1979), and the earlier work of Chomsky (1959; 1963), Miller and Chomsky (1963) and Chomsky and Miller (1963).

A grammar is a way of specifying a language. A language is a set of strings of symbols. A string, or sentence, is a finite ordered sequence of symbols. A symbol is a distinguishable object used in constructing the strings of a language. The alphabet, or terminal vocabulary, of a language is the finite set of all symbols which can appear in the strings of a language.

A language may have an infinite number of sentences in it. Such a language cannot be described by listing the sentences. Such a language can be described by giving a grammar for it. A grammar is a finite set of rules that specifies all the sentences in a language (Chomsky & Miller, 1963). Grammars use two types of symbol. Terminal symbols are the only ones that can appear in the strings of the language. Nonterminal symbols are used to denote intermediate constructions in the derivation of a sentence. The rules in a grammar are rewriting rules which tell how to rewrite a string of symbols to make another string of symbols. The specification, or "generation" starts with a "start" nonterminal symbol and proceeds at least until there are no more nonterminal symbols.

For example, the grammar given below generates the language which comprises the strings a, aa, aaa, aaaa, ... etc (Chomsky & Miller, 1963).

```
1       S  -->  a S
2       S  -->  a
```

S is a nonterminal symbol, the start symbol. a is a terminal symbol. The sentence aaaa is obtained by applying the rule 1 three times, and rule 2 once.

Different sorts of grammar specify or generate different sorts of language. The Chomsky hierarchy of grammars (Chomsky, 1959) contains the grammars discussed in this chapter. The Chomsky hierarchy is shown in Figure VIII-1. A language generated by a type i grammar is called a type i language.

Figure VIII-1 The Chomsky hierarchy of grammars

The Figure shows Chomsky's (1959) hierarchy of grammars. A hierarchy with more levels is given by Chomsky (1963). The Chomsky (1959) hierarchy is most commonly used (see Hopcroft & Ullman, 1979; Kain, 1972).

Rewriting rules for the grammars all have the form

$$\phi \dashrightarrow \theta$$

where ϕ and θ are strings of symbols.

Type	rewriting rules	comments
0	unrestricted	UTM equivalent
1	$\phi = X_1 A X_2$ and $\theta = X_1 w X_2$, where A is a single symbol, a nonterminal by convention, X_1 and X_2 are strings, and w is non-null.	called context-sensitive
2	ϕ is a single nonterminal symbol θ is non-null	called context-free
3	ϕ is a single nonterminal symbol θ is either a single terminal symbol, or a single terminal symbol followed by a single nonterminal symbol	finite automaton equivalent or regular

Now "... unrestricted rewriting systems are universal. If a language can be generated at all by what in the intuitive sense is a finitely stable well-defined procedure, it can be generated by a grammar of this type." (p.359 Chomsky, 1963). Chomsky goes on to say that linguists are

more interested in systems with less "generative power" than the type 0, or unrestricted rewriting systems. This is because unrestricted rewriting systems are thought to be more powerful than necessary for generating sentences of natural language.

VIII.2 UNIVERSAL TURING MACHINES

In section 2.1 the relationship between machines and languages is explained. In section 2.2 UTMs are explained.

VIII.2.1 Relationship between machines and languages

Machines may be compared to each other in terms of their ability to determine whether a sentence is a member of a language or not. The sentence is inputed to the machine, which may (a) give an output to say the sentence is a member of the language, which is called "accepting" the sentence, (b) give an output to say the sentence is not a member of the language---the sentence is not accepted---which may be called "rejecting" the sentence, or (c) never stop to indicate acceptance or rejection of the sentence, which may also be considered as rejection. A machine "accepts a language" if it accepts all the sentences in the language, and rejects all the sentences that are not in the language.

VIII.2.2 Universal Turing Machines (UTMs)

Turing machines (TMs. Turing, 1936), in particular universal TMs (UTMs) are briefly introduced in this section. The reader is referred to standard texts on the subject for more detailed descriptions; for example, Minsky (1967), Kain (1972) and Hopcroft and Ullman (1979).

A TM comprises a finite controller connected to an unbounded tape memory. The controller can move along the tape, reading symbols from it and writing symbols on it.

There are TMs that can accept all the languages generated by grammars in the Chomsky hierarchy: "In fact, any Turing machine can be

represented directly as an unrestricted rewriting system, and conversely." (p.357 Chomsky, 1963).

It turns out that there is a TM, the UTM, that, given a suitable description of any other TM, can "compute the same function" as the other TM (Minsky, 1967). Given on its tape both a description of any other TM, and an input sentence for that TM, the UTM will accept/reject the contents of its tape just in the case the other TM would accept/reject the sentence.

I show in section 3 that an MCLS can learn to simulate a UTM, a finite tape and the initial contents of the tape. Therefore the MCLS can learn any TM description that will fit on its tape. It can learn a direct representation of any unrestricted rewriting system, that will fit on its tape. It can learn any grammar that will fit on its tape.

Now a TM cannot be realized by any real machine, because real machines are finite and therefore cannot provide the infinite tape memory for a TM. However, a finite device may realize the finite controller of a TM. Such a finite device is said to have the competence of the TM whose controller it realizes, but not the performance ability of the TM (Nelson, 1978). Miller and Chomsky (1963) point out that

> One must be careful not to obscure the fundamental difference between, on the one hand, a device M storing the rules G but having enough computing space to understand in the manner of G only a certain proper subset L' of the set L of sentences generated by G and, on the other hand, a device M* designed specifically to understand only the sentences of L' in the manner of G. (p.467).

The competence of a TM which is realized by a UTM is contained in the rules for the UTM controller plus the finite description of the TM which is on the UTM's tape. The rules for the controller of the TM are represented by the description of it and the UTM's rules for interpreting the description.

Andreae and Cleary (1976) describe an MCLS that can learn to be a UTM controller using a tape in the outside environment. It is shown by Andreae and Cleary's paper that an MCLS can learn to simulate any finite state machine. The minimal MCLS to do this needs to have only one rule and must be able to do auxiliary actions in order to "remind" itself what it is doing. [The rules of MCLSs are explained in section 2 of chapter IV, and have nothing to do with rewriting rules.] These auxiliary actions can be separated out from the other actions by adding another rule to make an MCLS with two contexts both in short term memory (STM), and for making productions. Thus an MCLS can perform TM computations, given a "tape" in its environment which it can read from, write on and move along. Minsky's (1967) 7-state finite state controller for a UTM was simulated by the MCLS in Andreae and Cleary's paper. The infinite tape of this UTM was the environment. So the competence in the rules on the tape was not contained in the MCLS. However, the MCLS did have the performance ability of a TM since the behaviour of the TM did emerge from the MCLS. Hayes (1977) noted this in his criticism of the MCLS, "PURR-PUSS". Webb (1980) explains that the finite state part of a UTM, which he calls U, can use the environment as a TM tape:

> Embedding U in the universe introduces memory into it ... there is for U a distinction between its internal state and what it does, and its states constitute an internal memory in that they depend on previous states and inputs. But these 'inputs' may be outputs of U itself which were stored in U's environment, it's so called 'tape', which thus constitutes an external memory. (pp.14-15)

So the competence of a grammar implemented by the UTM in Andreae and Cleary (1976) is on the tape in the outside environment. In the simulation in section 3 the grammar is on tape inside the MCLS, so the MCLS has the competence of the grammar inside it.

If my TM-competent MCLS were to have access to the environment, like the MCLS of Andreae and Cleary (1976) that had a tape environment, then it would have the performance ability of a TM.

An early report (Andreae, 1974) established that an MCLS can carry out a task requiring more competence than that of a finite automaton. The task was counting.

Anderson (1976) has demonstrated a production system that simulates a TM, using its productions to handle the TM tape, as well as to implement the finite controller. An MCLS is a production system whose productions are formed from its stimuli and actions, as explained in section 2 of chapter IV. My demonstration in this chapter shows more than Anderson's demonstration since: (a) the productions have only actions and stimuli in them, and (b) all the productions, for the tape, finite-state controller, tape moving and writing, and tape remembering, must be acquired from the behaviour of the MCLS. That is, the MCLS must learn to be a TM.

VIII.3 AN MCLS THAT CAN SIMULATE A TURING MACHINE

The UTM-simulating MCLS (UTM-MCLS) simulates the seven state, four symbol UTM on page 279 of Minsky (1967). It can move along the tape, moving either one square to the left or one square to the right.

The UTM-MCLS learns to perform six operations:

moving along the tape
changing state
writing a symbol
reading a symbol
setting up a state change
setting up a symbol to write

For simulating the finite controller of the UTM the MCLS has 84 productions, which are equivalent to the 28 quintuples given by Minsky

(p.279). The quintuples specify the behaviour of the finite controller. They have the form

state, symbol --> state, symbol, move along tape

There are three productions per quintuple, one for predicting the new state, one for predicting the symbol to be written on the tape, and one for predicting the movement along the tape.

The UTM-MCLS does the six operations sequentially, one at a time. The state changing and symbol writing are both done in two steps by UTM-MCLS. The first step for each is a setting up step, as indicated in the list above. The new state and new symbol are held temporarily in intermediate actions in the short term memory (STM) of the MCLS until the old state and symbol have completed setting up the new symbol and state, and moved the controller along the tape.

The tape is simulated by two sets of productions in the long term memory (LTM) of the UTM-MCLS:

(i) a number of productions like "square 1, move right --> square 2", "square 2, move left --> square 1", and so on. There are two productions for each tape square. These productions represent the physical connectivity of a TM tape; and

(ii) a number of productions like "square 1 --> symbol P", and "square 22 --> symbol Y", one production for each tape square. These productions remember the contents of the tape. They might be called "pidgeon-hole" productions. For example, "square 22" is like a pidgeon hole, into which a symbol, say Y, may be put.

The UTM-MCLS is explained in detail in the appendix. The teaching required for the MCLS to learn to simulate the UTM is also explained.

VIII.4 RESTRICTIONS?

Now, the UTM-MCLS of section 3 has some trouble implementing the TM tape internally. There are three problems.

First, a general limitation of MCLS's: an MCLS has a finite set of possible actions and stimuli so that only a finite number of productions can ever be formed. Thus an MCLS could not use an unbounded amount of memory internally, even if it could be given that. Once the rules, possible actions and possible stimuli are determined, the maximum amount of memory that could be used by the MCLS is also determined. I shall call this memory limitation the "multiple context limitation" of the MCLS.

Now the second problem with an MCLS handling an unbounded amount of tape is this. If the MCLS is going to be able to use an unbounded amount of tape then it must be able to handle numbers of unbounded size so that it can use all the possible tape positions. These unbounded numbers must be held in STM for the tape number and tape moving contexts and must go into productions in LTM. However, any real MCLS has a finite STM. Minsky (1967) mentions this problem in connection with computers in general:

> The modern digital computer has a different memory structure---a large number of separately-accessible, finite-capacity "registers" or memory units. Of course, any real computer is actually finite, though the number of such registers may be very large. Our first thought might be to make our model like this, except with an infinite set of such registers. The trouble is that this would bring us back to something along the lines of a tape-like succession of the registers; we would need to have an infinite set of names or symbols for the registers, and this could not be handled directly by the finite-state part of the machine. (page 200)

This is exactly the problem---STM cannot hold an unbounded size number.

The third problem is to do with the learning of the productions for the tape moving rule. This problem is specific to my UTM-MCLS. It is overcome in the MCLS described by Andreae (1980a), and MacDonald and Andreae (1981). If the UTM-MCLS described in this chapter is to handle an unbounded length of tape, or at least a very long tape, then the number of productions that are stored for the tape moving rule is unbounded, or at least very large. In the UTM-MCLS two tape moving productions must be stored for every position on the tape, not just the part of the tape with input symbols on it. The problem is that it would take an unbounded, or very long, time for the MCLS to learn an unbounded, or very large, number of tape positions. Would the MCLS ever get to using the tape?

Now, if the MCLS is in a robot which has only a finite lifetime, then the first two problems disappear, because the MCLS can plenty of tape for a 70 year lifetime, as explained in sect below. The third problem, that of learning the tape product treated in two ways, as I explain in section 4.2 below: (a) productions should be in the MCLS before its "life" starts; UTM-MCLS is equivalent to a grammar between type 1 and type length of the tape learned is a linear function of the "inp length.

VIII.4.1 Finite lifetime

If an MCLS has a finite lifetime and is given only enough tape to do anything that it could do in its lifetime, it will have all the tape that it will ever need. That is, once the MCLS is restricted to having a finite lifetime, it needs only a finite tape. The length of the tape that it needs is no longer than the length that it would need if it spent all of its life moving in one direction along the tape. The restriction of having a finite lifetime is one I am happy to make, for

two reasons. Firstly, humans have a finite lifetime so giving an MCLS a finite lifetime does not prevent it from doing anything that humans can do. Secondly, there are probably very good practical reasons, for example economic reasons, for not planning for a robot to have a lifetime that isn't finite.[3]

Now, if an MCLS never runs out of tape during its lifetime then it never meets its multiple context limitation. That is, in the UTM simulation, for example, some of the numbers that STM can hold and that can be put into productions, never occur. Thus the UTM-MCLS generally won't reach its multiple context limitation because its finite lifetime will run out first. There is a good reason for having the MCLS never reach its multiple context limitation, apart from enabling the MCLS to handle a tape that is long enough that it never runs out in the MCLS's lifetime. After reaching its multiple context limitation an MCLS could not learn any productions with new contexts. It is reasonable that a robot should be able to learn to do new tasks, and learn to do tasks in new situations, over its entire useful lifetime. Therefore the MCLS must be designed so that it cannot reach its multiple context limitation during a lifetime of 70 years.

How much STM is required if the MCLS is to handle a reasonable lifetime's length of tape? To make an order of magnitude calculation, let us say that the lifetime expectancy of a robot with an MCLS in it is about 70 years. Say the MCLS makes a decision every second. In 70 years there are about 10^9 seconds. Then, if the MCLS moved along its tape continuously for 70 years in one direction, it would move past about 10^9 squares. This number must be doubled because the MCLS may go in either direction. In order to refer to 2×10^9 tape positions uniquely, the MCLS must be able to handle tape position numbers up to the number 2×10^9. Now 2×10^9 is about 2^{31}, so about 31 binary digits, or bits, are needed in STM to hold the tape position number.

This corresponds to an STM context only about five or six characters long! Most MCLS's used so far have productions with contexts that are this long, or longer (see Andreae, 1977a; Andreae, 1972-1983).

Now consider an MCLS that made a decision every tenth of a second. 35 binary digits would be needed. Even if the MCLS needed 10^{40} squares that would mean only about 130 binary digits. Therefore an MCLS can probably have a unique symbol for any useful length of tape.

VIII.4.2 Learning the tape

Although it seems reasonable for the UTM-MCLS to have a number as big as 10^9 or 10^{40} in STM, it doesn't seem so reasonable that there should be this many productions put into LTM, for moving up and down the tape, as a result of learning. The tape moving productions that would have to be learned by the UTM-MCLS to do the moving along a 2×10^9 square tape would number 4×10^9! Now, if the UTM-MCLS does only 10^9 or so decisions in its lifetime, it might not have time to learn to move along all the tape it might need. This problem is avoided in the parallel UTM-simulating MCLS in Andreae (1980a), and MacDonald and Andreae (1981). The counting is done as a multiple digit modulo-N count. It is necessary only to teach the counting of digits and carries, not every combination of digits. This means that for a tape of L positions one need teach only a number of productions much fewer than L.

My UTM-MCLS is probably able to accept a class of languages that lies somewhere between the context-sensitive class of languages and the context-free class of languages, without having to learn an impossible number of tape productions. Now, the class of context-sensitive languages is the class of languages generated by type 1 grammars. The class of context-free languages is the class of languages generated by type 2 grammars. The Chomsky hierarchy of grammars is shown in Figure VIII-1. All type 2 grammars are also type 1 grammars, but not

all type 1 grammars are type 2 grammars. All context-free languages are also context-sensitive languages, but not all context-sensitive languages are context-free. A TM that uses a tape only k times as long as its input sequence, where k is a constant, is a linear bounded automaton (LBA. Kain, 1972). So the UTM-MCLS can simulate an LBA by using a tape only k times as long as its input sequence.[4] An LBA can accept all context-free languages, and can probably accept some languages that are not context-free (Kain, 1972). The UTM need learn an amount of tape that is only a linear function of its input sequence. I would expect the input sequences to be extremely small in comparison to the UTM-MCLS's 70 year lifetime. For example, sentences in natural languages are that small. Unless k is extremely large, the number of tape squares required will be much smaller than the number of decisions the UTM-MCLS makes in a lifetime. The UTM-MCLS has to learn only 2k productions for every tape symbol it gets in the input sequence, one to move left and one to move right from each tape square needed.

Now a TM starts with a ready made tape. However, both my UTM-MCLS and Andreae's UTM-simulating MCLS learn the tape. The MCLSs built their own tapes. Perhaps it would be reasonable for the MCLSs to have the tape productions in LTM before their lifetimes start.

A TM tape is a sequence of physically connected memory units. A finite unit, the read/write/move head of the TM controller of the TM "remembers" the unique place where the finite-state controller is on the tape by being there. This is possible because the tape squares are physically connected in sequence. In a digital computer or an MCLS, however, memory is not like this. The only way to remember which memory unit or production was last "read from" is to hold a unique symbol, the "address", for that memory unit or production. This sort of remembering cannot be done by a finite device (Minsky, 1967, p.200). The way the physical connectivity of the TM tape can be simulated is by having a

set of productions, the counting productions, learned. The productions say which unique symbol, or number, is the address of the memory unit on each side of each memory unit. Perhaps the productions representing the physical connectivity of the tape should be in the MCLS at the beginning of its lifetime.

VIII.5 TEACHING WITH REFLEXES OR LEADING

During the teaching in the appendix the teacher selects the actions for the UTM-MCLS. He guided the MCLS. Guiding is discussed in section 3.2 of chapter II. Reflexes or leading could also have been used in the teaching.

Suppose there is a separate reflex set up for each action in the UTM simulation. Suppose that the reflexes are triggered by stimuli which the teacher can cause. Then to evoke an action in the MCLS the teacher need only cause the corresponding stimulus to occur. Since there are no contexts with stimuli in them, the stimuli do not affect the MCLS's decisions. For example, the teacher might have a set of buttons, each of which had an action written on it. The buttons might cause the reflexes. This is little different from the teacher typing in actions directly, as in the appendix. It is likely that the teacher would need to refrain from causing reflexes while the robot selected actions.

The teaching could also be achieved using the leading method. Suppose that there is an action that the teacher can lead for every action in the UTM simulation. Then as long as the teacher can lead the actions in the sequences required in the simulation, he can teach the required sequences. Since there are no stimuli in the contexts, these actions could have any external effects at all. The stimuli they caused would not affect the MCLS.

VIII.6 'GRAMMAR' LIKE PROPERTIES OF THE UTM SIMULATION

Are there any grammar-like properties in the actual simulation of a UTM by the MCLS? Consider the sequence,

... //S1 IWRITE **Q IMOVE1 ML ISLASH SS1 IASTERISK YQ ...

from the UTM simulation in Figure VIII-8 of the appendix.

Note how:

(a) the identifier rule productions carry out a syntactical function, organizing actions that do things by interspersing "function" or identifier actions;

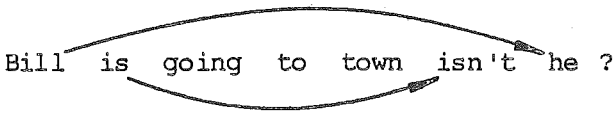
(b) the slash and asterisk contexts in STM hold information that must be used later in the sequence when the identifier rule productions signal for it.

Without the identifier actions the sequence above is

or if the simulation were to be altered so that the actual symbol writing preceded the state change instead of following it,

The identifier actions might have no effects on a robot's body that would be seen by someone interacting with the robot. They might be subliminal actions. The identifier actions might be thought of as nonterminal symbols, since (i) they need not appear in the observable action sequence, and (ii) they have only a syntactic function.

Compare the sequence immediately above with the sentence,



Here also certain words have information about them held until later in the sequence, when the tag question is formed (see Kuiper, 1980a). I am

not trying to draw a parallel here between the UTM-MCLS and tag question formation. I am pointing out that an MCLS can hold information (e.g. //S1) and use it at a specific point in a sequence (e.g. to do the SS1 after ISLASH).

Work underway is intended to produce a design for an MCLS that will do some of the processes ascribed to humans when they use natural language (Andreae, 1978; 1979a; 1979b). In particular, Andreae (1979a) has described an MCLS which performs a task with about two levels of "self-embedding" in its behaviour [The sentence "The man the boy the dog loves saw ran" has two relative clauses self-embedded in the main sentence (Brown, 1973). It is difficult for us to understand, even when written down.].

VIII.7 WHAT DOES AN MCLS'S BEING ABLE TO LEARN ANY GRAMMAR SHOW?

Chomsky and Miller (1963) state:

> The fundamental fact that must be faced in any investigation of language and linguistic behaviour is the following: a native speaker of a language has the ability to comprehend an immense number of sentences that he has never previously heard and to produce, on the appropriate occasion, novel utterances that are similarly understandable to other native speakers. The basic questions that must be asked are the following:
>
> 1. What is the precise nature of this ability?
> 2. How is it put to use?
> 3. How does it arise in the individual? (p.271)

What this chapter has shown---in showing that an MCLS can learn any grammar that fits into its memory---is that MCLSs do not lack the ability to acquire and implement the structures that have been proposed by linguists in response to the first question. Briefly, linguistics propose that the precise nature of a native speaker's ability is characterized by some, yet to be determined, class of grammars that is

a restricted subset of the Chomsky hierarchy (Wexler & Culicover, 1980; Berwick & Weinberg, 1982; Chomsky, 1963, Chomsky, 1965).

The sort of MCLS that is required for implementing language in a way that (a) enables a language to be acquired, and (b) enables language to be used for communicating, is a different and, to me, more important concern.

For example, just because an MCLS in a robot can simulate a UTM does not make it teachable or adaptable. The UTM-MCLS hardly interacts with its environment at all. It just receives sequences typed in by a teacher; a very limited interaction with the teacher. Even for an MCLS to use a TM tape in the environment, all that is required is that the robot's actions be stored in the environment and, later, be sensed by the robot (Andreae & Cleary, 1976).

Just because an MCLS can learn any grammar in the way shown in section 3 does not mean that it can learn a grammar of natural language given the sort of information available to, for example, a child.

Just because an MCLS can implement any grammar does not mean that it can communicate with language in real time. For example the implementation of a grammar of natural language in a robot should enable the robot to process sentences rapidly enough for it to have a conversation with a human. It has been suggested that this sort of constraint can be used to rule out some grammars as grammars for natural language (Berwick & Weinberg, 1982).

In order to use language, more is required than grammar rules and word meanings. Knowledge about the real world and about the interactive communication process is also required (Oim, 1981).

> ... linguistic communication constitutes a specific kind of goal-oriented social activity regulated by certain norms and other conventions. (p.69 Oim, 1981)

VIII.8 DOES AN MCLS NEED TO BE ABLE TO LEARN EVERY GRAMMAR?

The terms "grammar" and "algorithm" may be used interchangeably, since (a) a type 0 grammar is equivalent to a Turing machine (see section 2), and (b) it is generally agreed that a Turing machine can implement any algorithm, that is, any effective procedure (see the start of this chapter).

An MCLS will not need to be able to learn every grammar, or algorithm, that would fit into its memory. There will be various reasons for ruling out algorithms as algorithms that an MCLS must be able to learn.

Firstly, algorithms that are over a certain length would not need to be learned by an MCLS. If the sheer length of an algorithm required an MCLS to devote more than, say, half of its lifetime to learning that algorithm, then it is reasonable to suppose that the algorithm need never be acquired.

Secondly, if a native speaker's ability can be characterized by a restricted subset of the Chomsky hierarchy, and an MCLS can in practice learn, and in practice implement grammars in this subset, then it would indeed be true that MCLSs did not need to be able to learn every grammar in order to learn language. However, that is not to say that an MCLS wouldn't need to learn to other algorithms for other reasons. A robot might be expected to learn grammars for computer languages.

What algorithms does an MCLS need to be able to learn? Well, what if an MCLS could not learn rules above a particular level in the Chomsky hierarchy? This would limit its performance ability only under two conditions.

Firstly, if these rules had to be used with an unbounded working memory in the environment, then no lower level rule could produce the same performance. The MCLS wouldn't have the performance ability required. The behaviour of any type of rule in the Chomsky hierarcy

that is used with only a finite amount of working memory, can always be mimicked by a finite automaton (Nelson, 1978). The rule-plus-finite-memory is finite. So any rule can be mimicked by a finite automaton if only a finite working memory is required.

Secondly, other limitations of the MCLS might prevent lower level implementations of the rules. Even if only a finite working memory is used, a lower level implementation would not be possible. For example, a finite automaton equivalent to a particular rule-plus-finite-memory might take too long to acquire in practice, while a higher level rule might not.

Whether an MCLS finds it easier to learn a type 0, 1, 2 or 3 rule is not certain. Neither is it certain how much working memory is required for the rules that are learned. However, it is known, for example, that self-embedded relative clauses in normal spoken language go to only about two levels (Brown, 1973. See the sentence given at the end of section 6.). So not much working memory is required, compared to the huge working memory an MCLS can have inside it, as explained in section 4.1.

Note that an MCLS can hardly help having the performance ability of a TM. Only a very simple MCLS is required for simulating a TM's behaviour, if it can use a tape in the environment (Andreae & Cleary, 1976). Only one rule is required in the MCLS. The simulation given in Andreae and Cleary (1976) is discussed in section 2. A robot that could use pen and paper would have an unbounded memory available in the environment.

VIII.9 CONCLUSION

I have demonstrated that an MCLS can learn any grammar that will fit into its finite memory. Hayes (1977) asserted that the MCLS PURR-PUSS "... certainly couldn't learn a grammar for any reasonably interesting subset of English.".

The finite memory of an MCLS can be seen as not being a limitation for two reasons: (a) the memory is large enough to be indistinguishable from an unbounded memory if the robot's lifetime is about 70 years long, and (b) an MCLS can use the environment as an unbounded memory.

The simulation of a UTM by the MCLS has not required anything to be said about stimuli coming from the environment, nor about any action having an effect on the environment. Showing that an MCLS can learn grammars takes no account of the requirement that an MCLS do things in the environment that enable it to learn, survive, interact with other things, et cetera.

NOTES FOR CHAPTER VIII

1. What an MCLS does learn will never actually be constrained by what can fit in the MCLS's memory. I point out in section 4.1 that an MCLS for a real robot will have enough memory to last for all of a robot lifetime of 70 years. Rather than a constraint on what can fit in an MCLS's memory, an actual constraint on something that an MCLS might learn would be: Can it be learned in a small fraction of the robot's lifetime? As I point out in section 7, questions to do with the actual acquisition and use of a grammar by an MCLS are not the concern of this chapter. The concern is to show that an MCLS can in principle learn grammars of any level in the Chomsky hierarchy. The Chomsky hierarchy of grammars is given in section 1.

2. Schubert (1978) criticised the MCLS PURR-PUSS on the grounds that it could not generalize. That this is incorrect is shown in chapters VII and IX. I show in chapter VII that reflex actions, actions automatically triggered by stimuli, can be generalized to situations where the reflex-trigger is not present. I demonstrate in chapter IX that an MCLS can handle a specific problem of generalization called negation. Something positive is done when a condition like "not-x" occurs, without introducing a specific symbol for not-x or teaching the MCLS what to do when each member of the rest of the universe set occurs. Andreae (1977a) has explained that the MCLS PURR-PUSS can generalize, and has refuted Schubert's criticism (1979c).

3. Anderson (1976) says:

> It may be the case that human memory is so large that it would not be exhausted in a lifetime of simulating the TM. (p.143)

4. The constant k can always be made 1, but to do this the number of symbols must be increased (Hopcroft & Ullman, 1979). The UTM-MCLS would have to learn more transition productions, more identifier productions and more asterisk productions, if more symbols were used.

APPENDIX FOR CHAPTER VIII

UNIVERSAL TURING MACHINE SIMULATION BY AN MCLS

In section A.1 the UTM-simulating MCLS (UTM-MCLS) is described. In section A.2 the operation of the simulation is described. The teaching required for the MCLS to simulate the UTM is described in section A.3.

VIII.A.1 MCLS description

The rules and templates of MCLSs are explained in section 2 of chapter IV. The actions that the UTM-MCLS performs are shown in Figure VIII-2. A context of the UTM-MCLS will be written as a string comprising the actions in the context, concatenated. For example, a context with the identifier action IMOVE1 and the move action MR in it is written IMOVE1MR. There will be no ambiguity as to what the actions in a concatenated string are. Productions will be written in the form: context --> prediction. For example IMOVE1MR --> ISLASH is a production.

The production templates and priorities for each rule of the UTM-MCLS are shown in Figure VIII-3. The highest priority predicted action is performed. The highest priority possible is 1. In the UTM simulation there is no conflict between the several priority 2 rules. In the UTM-MCLS, productions can have only one prediction.

Briefly revising the operation of an MCLS from section 2 of chapter IV, the production templates form productions in long term memory (LTM) using the contexts, which are given by each context template as actions arrive, and the predicted actions, which are given by the prediction templates. The present short term memory (STM) contents, the contexts in the multiple context (MC), are compared with previous contexts, the contexts in LTM. An action to perform is

Figure VIII-2 Actions of UTM-MCLS

(a) Types of action

Type of Action	First Character	Function
Identifier	I	Syntactical
Slash (or /)	/	Holds new state
State	S	State
Asterisk (or *)	*	Holds new symbol
Symbol	Y	Symbol
Number	N	Tape position number
Move	M	Direction of movement along tape.

(b) Actions used by UTM-MCLS

Type	Action	Comments
Identifier		A different identifier action precedes each of the six different operations of the UTM-MCLS
		Operation following action
	IMOVE1	move along tape, either left or right
	ISLASH	change state of UTM
	IASTERISK	write symbol on the tape
	IMOVE2	read symbol from the tape
	ISTATE	set up next state change
	IWRITE	set up next symbol to write
Slash (/)	//S1,//S2,//S3,//S4,//S5,//S6,//S7	action for temporary storage of next state change
State	SS1,SS2,SS3,SS4,SS5,SS6,SS7	actual state of UTM; one of seven
Asterisk (*)	**P,**Q,**A,**Y	action for temporary storage of next symbol to be written on tape
Symbol	YP,YQ,YA,YY	actual symbol actions; P, Q, A, and Y
Number	N0,N1,N2,N3, ...	tape square number of UTM tape
Move	MR,ML	action for moving along the tape, either one square to the left (ML), or one square to the right (MR)

Figure VIII-3 The production templates and priorities of UTM-MCLS rules

Identifier rule:
context template: last identifier and last action
prediction template: identifier actions
priority: 1
example production: IMOVE1MR --> ISLASH
function in simulation: organizes six operations

Slash rule:
context template: last action if it was an identifier and last /action
prediction template: state actions
priority: 2
e.g. ISLASH//S1 --> SS1
function: changes state

Asterisk rule:
context template: last action if it was an identifier and last *action
prediction template: symbol actions
priority: 2
e.g. IASTERISK**P --> YP
function: writes on tape

Tape move rule:
context template: last action if it was an identifier and last move action and last number action
prediction template: number actions
priority: 2
e.g. IMOVE2N1ML --> N0
function: moves along tape

Transition rule:
context template: last action if it was an identifier and last state action and last symbol action
prediction template: move, / and * actions
priority: 2
e.g. IMOVE1SS1YP --> ML
function: sets up state change, symbol to write, direction to move along tape

Tape rule:
context template: last number action
prediction template: symbol actions
priority: 3
e.g. N1 --> YP
function: remembers symbols on each tape square

selected from the set of actions predicted by the LTM contexts which match the STM contexts. If there are no matching contexts, and hence no predictions, then the person interacting with the MCLS is given the chance to put an action in. It is stored in productions, and put into contexts, just as any action performed by the MCLS would be.

Some comments about the six rules and their part in the UTM simulation are made below. Also, the example UTM-MCLS operation shown in Figure VIII-4 is explained.

The productions needed in LTM for the MCLS to perform all the six UTM operations are shown in Figure VIII-5. The six UTM operations are:-

moving along the tape
changing state
writing a symbol
reading a symbol
setting up the state change
setting up the symbol to be written,

as shown in Figure VIII-2.

As an example, Figure VIII-4 shows the MCLS when the STM has this MC in it:-

contexts	rule
IMOVE1 IMOVE1	identifier
IMOVE1 //S1	slash
IMOVE1 **Q	asterisk
IMOVE1 SS2YY	transition
IMOVE1 N22MR	move
N22	tape

This is the MC following the first IMOVE1 action in Figure VIII-8.

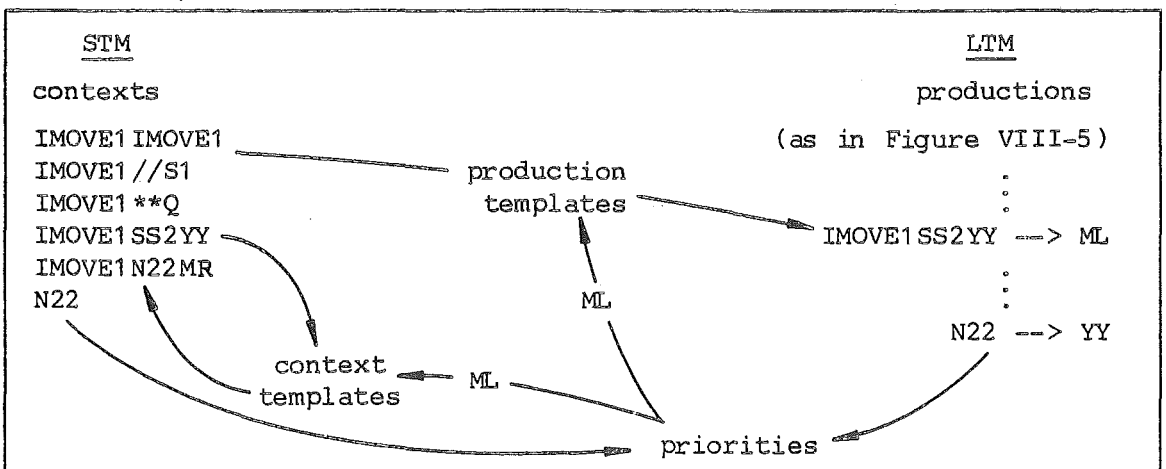


Figure VIII-4 The Operations of an MCLS that simulates a UTM

Identifier productions

```
IMOVE1       MR,ML                                  --->  ISLASH
ISLASH       SS1,SS2,SS3,SS4,SS5,SS6,SS7            --->  IASTERISK
IASTERISK    YP,YQ,YA,YY                            --->  IMOVE2
IMOVE2       YP,YQ,YA,YY                            --->  ISTATE
ISTATE       //S1,//S2,//S3,//S4,//S5,//S6,//S7    --->  IWRITE
IWRITE       **P,**Q,**A,**Y                        --->  IMOVE1
```

Slash productions

```
ISLASH  //S1  --->  SS1
ISLASH  //S2  --->  SS2
  ..     ..          ..
  ..     ..          ..
ISLASH  //S7  --->  SS7
```

Asterisk productions

```
IASTERISK  **P  --->  YP
IASTERISK  **Q  --->  YQ
IASTERISK  **A  --->  YA
IASTERISK  **Y  --->  YY
```

Tape move productions

```
IMOVE2  N1  ML  --->  N0        IMOVE2  N0  MR  --->  N1
IMOVE2  N2  ML  --->  N1        IMOVE2  N1  MR  --->  N2
IMOVE2  N3  ML  --->  N2        IMOVE2  N2  MR  --->  N3
IMOVE2  N4  ML  --->  N3        IMOVE2  N3  MR  --->  N4
IMOVE2  N5  ML  --->  N4        IMOVE2  N4  MR  --->  N5
IMOVE2  ..  ML  --->  ..        IMOVE2  ..  MR  --->  ..
```

[Note: N0,N1,N2,N3,N4 and N5 are six consecutive tape square numbers.]

Transition productions

```
IMOVE1 SS1 YP  --->  ML
ISTATE SS1 YP  --->  //S2
IWRITE SS1 YP  --->  **P
```

This is one of the 28 transitions. From state 1 and symbol P it causes a transition to state 2, a move left and no change to P. There are three transition productions for each of the 28 transitions. All 28 groups of three are shown in Figure VIII-6. Each three have the same form as this example, but with SS1, YP, ML, //S2 and **P replaced by the current state action, the current symbol action, the next move action, the /action for the next state and the *action for the next symbol, respectively.

Tape productions

```
N1  --->  YP      N10  --->  YQ      N20  --->  YP
N2  --->  YP      N11  --->  YQ      N21  --->  YP
N3  --->  YQ      N12  --->  YQ      N22  --->  YY
N4  --->  YP      N13  --->  YQ      N23  --->  YY
N5  --->  YQ      N14  --->  YP      N24  --->  YA
N6  --->  YP      N15  --->  YQ      N25  --->  YY
N7  --->  YP      N16  --->  YQ      N26  --->  YY
N8  --->  YP      N17  --->  YP      N27  --->  YA,
N9  --->  YQ      N18  --->  YP      N28  --->  YY
                  N19  --->  YQ      N29  --->  YY
                                     N30  --->  no prediction
```

Figure VIII-5 Productions that are needed for UTM-MCLS to perform TM operations on its own internal "tape". The tape symbol productions are immediately above. The UTM starts in state 2 at position 22 with these symbols on the tape. In this demonstration the tape symbols are put in as the MCLS "moves" to each square for the first time. In the Figure, a comma between two action means "or".

Figure VIII-6 All the transition productions for UTM-MCLS

The UTM transitions are given by Minsky (1967) on page 279.

```
IMOVE1 SS1 YY ---> ML  ISTATE SS1 YY ---> //S1  IWRITE SS1 YY ---> **Q
IMOVE1 SS1 YQ ---> ML  ISTATE SS1 YQ ---> //S1  IWRITE SS1 YQ ---> **Q
IMOVE1 SS1 YP ---> ML  ISTATE SS1 YP ---> //S2  IWRITE SS1 YP ---> **P
IMOVE1 SS1 YA ---> ML  ISTATE SS1 YA ---> //S1  IWRITE SS1 YA ---> **P

IMOVE1 SS2 YY ---> ML  ISTATE SS2 YY ---> //S1  IWRITE SS2 YY ---> **Q
IMOVE1 SS2 YQ ---> MR  ISTATE SS2 YQ ---> //S2  IWRITE SS2 YQ ---> **Y
IMOVE1 SS2 YP ---> MR  ISTATE SS2 YP ---> //S2  IWRITE SS2 YP ---> **A
IMOVE1 SS2 YA ---> MR  ISTATE SS2 YA ---> //S6  IWRITE SS2 YA ---> **Y

IMOVE1 SS3 YY ---> ML  ISTATE SS3 YY ---> //S3  IWRITE SS3 YY ---> **Y

IMOVE1 SS3 YQ ---> MHALT  UTM-MCLS will "halt" if state 3 is reached
ISTATE SS3 YQ ---> /HALT  and the symbol Q read. There are no contexts
IWRITE SS3 YQ ---> *HALT  in LTM with MHALT, /HALT or *HALT in them.

IMOVE1 SS3 YP ---> ML  ISTATE SS3 YP ---> //S3  IWRITE SS3 YP ---> **A
IMOVE1 SS3 YA ---> ML  ISTATE SS3 YA ---> //S4  IWRITE SS3 YA ---> **P

IMOVE1 SS4 YY ---> ML  ISTATE SS4 YY ---> //S4  IWRITE SS4 YY ---> **Y
IMOVE1 SS4 YQ ---> MR  ISTATE SS4 YQ ---> //S5  IWRITE SS4 YQ ---> **Y
IMOVE1 SS4 YP ---> ML  ISTATE SS4 YP ---> //S7  IWRITE SS4 YP ---> **P
IMOVE1 SS4 YA ---> ML  ISTATE SS4 YA ---> //S4  IWRITE SS4 YA ---> **P

IMOVE1 SS5 YY ---> MR  ISTATE SS5 YY ---> //S5  IWRITE SS5 YY ---> **Y
IMOVE1 SS5 YQ ---> ML  ISTATE SS5 YQ ---> //S3  IWRITE SS5 YQ ---> **Y
IMOVE1 SS5 YP ---> MR  ISTATE SS5 YP ---> //S5  IWRITE SS5 YP ---> **A
IMOVE1 SS5 YA ---> MR  ISTATE SS5 YA ---> //S5  IWRITE SS5 YA ---> **P

IMOVE1 SS6 YY ---> MR  ISTATE SS6 YY ---> //S6  IWRITE SS6 YY ---> **Y
IMOVE1 SS6 YQ ---> ML  ISTATE SS6 YQ ---> //S3  IWRITE SS6 YQ ---> **A
IMOVE1 SS6 YP ---> MR  ISTATE SS6 YP ---> //S6  IWRITE SS6 YP ---> **A
IMOVE1 SS6 YA ---> MR  ISTATE SS6 YA ---> //S6  IWRITE SS6 YA ---> **P

IMOVE1 SS7 YY ---> MR  ISTATE SS7 YY ---> //S7  IWRITE SS7 YY ---> **Q
IMOVE1 SS7 YQ ---> MR  ISTATE SS7 YQ ---> //S6  IWRITE SS7 YQ ---> **Y
IMOVE1 SS7 YP ---> MR  ISTATE SS7 YP ---> //S7  IWRITE SS7 YP ---> **P
IMOVE1 SS7 YA ---> MR  ISTATE SS7 YA ---> //S2  IWRITE SS7 YA ---> **Q
```

Note that the ML action, for "Move Right", in the move context was the last move action done during teaching and so isn't shown in Figure VIII-8 (b). The SS2 action for state was also done during teaching and isn't shown in Figure VIII-8 (b). If the productions in Figure VIII-5 are in LTM then, following the IMOVE1 action above, there will be predictions for actions ML and YY. The production

IMOVE1SS2YY --> ML

will have matched the transition context above and the production

N22 --> YY

will have matched the tape context above. The prediction for ML is a higher priority one, according to the priorities of Figure VIII-3, and so the ML is performed, as shown in Figure VIII-8.

The UTM that the UTM-MCLS simulates is the UTM on page 279 of Minsky (1967). It has seven states and uses four symbols on its tape. It can move one tape square each way, to the left or right. The finite controller is specified by a transition table with 28 transitions in it. A transition says which state to change to, which symbol to write, and which way to move along the tape, when a particular symbol is encountered while the finite controller is in a particular state. Here the symbols P and Q are used instead of 1 and 0, respectively.

The tape is stored in the tape productions and move productions. The tape productions are of the form

number --> symbol.

Thus, if a number action is performed, the tape context is matched with a production in LTM to give a prediction for the symbol action for that position on the tape. If no other productions are matched, then the symbol action is performed, which has the effect of putting that symbol in the transition context in STM. The tape has then been read into the transition context. Writing onto the tape is accomplished in two parts. The transition productions are the transition table of the controller

of the UTM. The transition productions cause an * symbol action, **A, **Y, **Q or **P, to be performed, which is saved in STM and causes a symbol action, YA, YY, YQ or YP, to be done later by the asterisk rule. The symbol action replaces the last symbol action predicted by the current tape context, thus making a new production in LTM for the "newly-written-on tape square". That is, the symbol predicted by the current tape square number, is replaced by the new symbol. It is important that the old production for a newly-written-on tape square is replaced by the new one. The tape rule is a recency rule. Only one prediction, the most recent one, is stored with a tape context. MCLSs designed before this one, which was first described in MacDonald (1980), had only choice rules. In a choice rule each new prediction is added. Old productions are not replaced by new ones.

The state change is also done in two steps, using the slash productions as an intermediate. It is necessary that the new state and symbol are in these intermediate actions in STM so that the current state and symbol can do all three jobs of state changing, symbol writing and moving along the tape. The tape moving action can be performed without an intermediate since it doesn't change the current state and symbol. The tape moving is done by the tape move productions which, given the last number action and last move action, do the next number up or down.

The identifier productions keep the six different operations separate and following one another in a specific order.

The UTM in Minsky's book starts in state 2 at a position 22 squares from the left-hand end of the tape. Figure VIII-7 shows how the identifier productions control all the six operations. Where there is more than one possible action, the diagram in Figure VIII-7 is annotated to show the rule concerned. Figure VIII-8 (b) shows a sample sequence of actions, beginning where Minsky's UTM starts, in state 2 at

Rule	Context and Action
Identifier	
Transition	state, symbol --> move
Identifier	
Slash	/state --> state
Identifier	
Asterisk	*symbol --> symbol
Identifier	
Move	number, move --> number
Tape	number --> symbol
Identifier	
Transition	state, symbol --> /state
Idendifier	
Transition	state, symbol --> *symbol
Identifier	

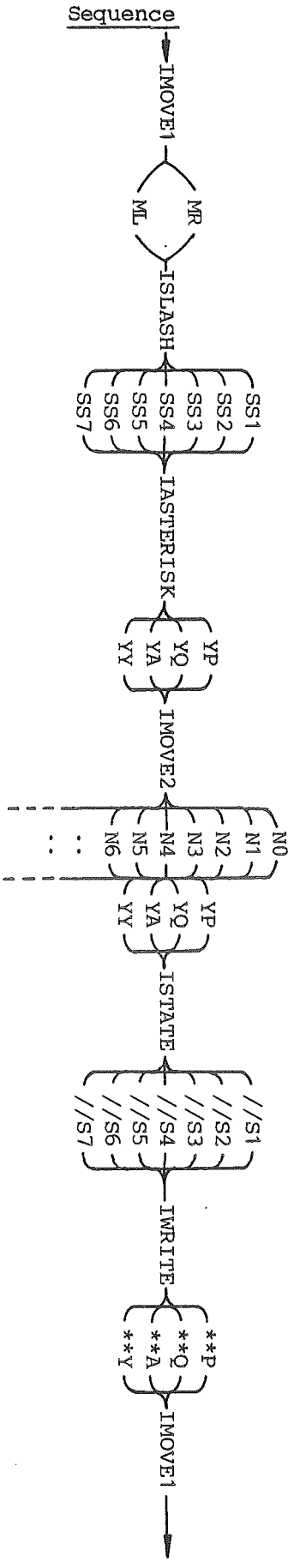


Figure VIII-7

The identifier rule orders the operation of the other rules: a syntactic function.

square 22. The teaching in Figures VIII-9 and VIII-10 puts the productions of Figure VIII-5, except for the tape productions, into LTM, thus enabling the MCLS to do the actions of Figure VIII-8 (b). The teaching is explained in section A.3. First I will explain in section A.2 how the MCLS simulates the UTM.

VIII.A.2 Operation of the simulation

The contents of the STM, before the first IMOVE2 action in Figure VIII-8 (b) is performed, are shown in Figure VIII-8 (a). The events shown in Figure VIII-8 (a) are in STM at the end of the teaching. This is explained in section A.3.

(a) The contents of STM before the sequence in (b). This is the MC at the end of the teaching

last identifier	IMOVE2
last action	MR
last /action	//S2
last *action	**Q
last state action	SS2
last symbol action	YQ
last number action	N21
last move action	MR

The only valid contexts are:

identifier context IMOVE2MR

tape context N21

because the last action was not an identifier.

(b) Sequence of events following MC in (a). The beginning of the UTM simulation.

IMOVE2 N22 YY ISTATE //S1 IWRITE **Q IMOVE1 ML ISLASH SS1 IASTERISK YQ IMOVE2 N21 YP ISTATE //S2 IWRITE **P IMOVE1 ML ISLASH SS2 IASTERISK YP IMOVE2 N20 YP ISTATE //S2 ...

Figure VIII-8

After the MC shown in Figure VIII-8 (a) the identifier rule performs IMOVE2. That is, the production

IMOVE2MR --> IMOVE2,

as learned in Figure VIII-10, is the highest priority matching production. The only STM change following the IMOVE2 action is that the

last action is now IMOVE2. Before the IMOVE2 in Figure VIII-8 (b) was performed the last action was MR. A move context of IMOVE2MRN21 predicts N22. N22 is performed, so now the last number action is N22. There are no predictions, that is no matching productions, for this MC. Thus the teacher has a chance to put in an action. He puts in the symbol action, YY, that is the symbol for the 22nd square of the tape. The teacher puts in a symbol action each time the MCLS comes to a position on the tape that it hasn't been to before. In this way, the symbols are taught as the MCLS goes along the tape. John Andreae's MCLS for simulating a UTM has the symbols put on its tape before the start of the UTM simulation (Andreae, 1980a; MacDonald & Andreae, 1981).

For the 22nd position, the symbol action is YY. Once YY is done, there is a prediction for ISTATE from the identifier production

IMOVE2YY --> ISTATE.

Once the ISTATE action is done, the MC is:-

rule	context
identifier	ISTATEISTATE
slash	ISTATE//S2
asterisk	ISTATE**Q
transition	ISTATESS2YY
move	ISTATEMRN22
tape	N22

Thus the transition production

ISTATESS2YY --> //S1

is matched and //S1 is performed. Later, the slash rule will do the SS1 action as a result of having the //S1, just performed here, in its context. This completes the change to state 1. The identifier production

ISTATE//S1 --> IWRITE

now causes IWRITE to be done. The transition context predicts **Q, which is performed. Next, identifier rule causes IMOVE1 and then transition rule causes ML. The MCLS has carried out the three parts of the transition from state 2 and symbol Y, changing to state 1, writing

symbol Q and moving left.

Next the identifier production,

IMOVE1ML --> ISLASH

causes ISLASH to be done. The MC is now,

rule	context
identifier	ISLASHISLASH
slash	ISLASH//S1
asterisk	ISLASH**Q
transition	ISLASHSS2YY
move	ISLASHMLN22
tape	N22

so that the highest priority predicted action, SS1, from slash productions in LTM, is performed. Identifier rule causes IASTERISK and asterisk production

IASTERISK**Q --> YQ

works through asterisk rule to cause the action YQ to be done. When YQ is performed the tape rule context template and the tape rule prediction template are matched so that the production N22 --> YY is changed to N22 --> YQ. This is how the MCLS writes on its tape. Identifier rule causes IMOVE2 again and the cycle continues.

The "UTM" continues on until it reaches position 29 in state 3 with the tape being

N17	N18	N19	N20	N21	N22	N23	N24	N25	N26	N27	N28	N29	N30
YA	YA	YY	YA	YA	YY	YY	YY	YY	YY	YP	YY	YY	YA

as on page 280 of Minsky's (1967) book. I consider, like Andreae and Cleary (1976), that this is sufficient to show that the MCLS is in fact simulating Minsky's UTM.

VIII.A.3 Teaching

It is not so important how actions are first performed by the UTM-MCLS. They are just "put in" by the teacher, the person interacting with the MCLS. For a real robot with an MCLS in it there are constraints on what actions and stimuli can occur and on the order in which they can occur. These constraints are imposed by the robot body,

the environment, and the MCLS itself. The constraints that are external to the MCLS are not the concern of this chapter. The concern of this chapter is the capability of the MCLS itself. I show that, in principle, an MCLS can learn to simulate a UTM, if such learning is available in the environment. Other ways for actions to be first performed, rather than being "put in" by a teacher, are by reflex, back-reflex, the leading method, and spontaneously, as explained in section 3 of chapter II, and in section 1.3 of chapter VII (see also MacDonald, 1979, 1980, 1981, 1982a; 1982b). It is explained in section 5 how the MCLS which learns to simulate the UTM could have been taught using either reflexes or the leading method. The nature of this explanation shows why the method of teaching is not so important. It shows how the UTM demonstration says little about how an MCLS interacts through a body with the world.

Now, in order for the UTM-MCLS to learn to simulate the UTM, the productions given in Figure VIII-5 must come to be in LTM as a result of learning. The UTM-MCLS has in fact been taught to do all the TM operations. The UTM-MCLS was implemented in BASIC on a TRS-80 microcomputer.

Another action is needed in order to teach the productions in Figure VIII-5. It is a dummy action that is none of the types previously mentioned in the specification of this MCLS. I used the action "ZZ". A particular state and symbol must be set up in the transition context in STM so that the transitions can be taught. This can be done by replacing the tape numbers in Figure VIII-8 by "ZZ" and by inserting "ZZ" between ISLASH and the state action, as well. The ZZ can always be put in those two places as long as it is there right from the start. Also, none of the rules that require a "last action an identifier action" in their contexts can store productions predicting an action which follows ZZ. ZZ is not an identifier. This turns out to

be useful in teaching: (i) a ZZ action after IMOVE2 and followed by a symbol action prevents asterisk rule storing a production there, and (ii) a ZZ after ISLASH and followed by a state action prevents slash rule storing a production that is not in Figure VIII-5. A third effect of the ZZ is that teacher can always put in an action of his or her choice after the ZZ since, in the teaching sequence, actions that follow ZZ never go into productions in LTM. If these actions never go into productions in LTM then they can never be predicted and performed in a similar situation later, so the teacher always has the chance to put in an action after the ZZ. Recall that the person interacting with the MCLS can put in actions only if none are predicted. The sequence would go like that in Figure VIII-9. It is explained in detail below.

ISLASH ZZ SS2 IASTERISK YP IMOVE2 ZZ YP ISTATE //S2 IWRITE **A IMOVE1 MR ISLASH ZZ SS3 IASTERISK YA IMOVE2 ZZ YA ISTATE //S4 IWRITE **P IMOVE1 ML ISLASH ... (and so on for all the transitions.)

Figure VIII-9 The teaching sequence: ZZ actions prevent the slash production ISLASH//S2 —> SS3 (which would give a false state change) being stored, for example.

Briefly, in Figure VIII-9 state 2 and symbol P are set up with a transition to state 2, writing symbol A and moving right one square on the tape. Also, the transition from state 3 with symbol A to state 4, writing a P and moving left, has been taught. Note that the slash, tape and tape moving rules are not being used at all here. The productions for these rules are taught later on. Note also that in Figure VIII-9 all the asterisk, transition and identifier productions are taught as the teaching goes along.

The sequence in Figure VIII-9 is the first sequence of actions for the MCLS. So, before the first action in the Figure, LTM and STM are

empty. There are no predictions at the start. Teacher puts in ISLASH and then ZZ. STM becomes:-

last action ZZ

last identifier ISLASH.

Note that ZZ goes only into the identifier context, as the last action. It goes into no other contexts. It is predicted by no contexts. The teacher then does SS2 to set up the transition context in state 2. The identifier context is ISLASHSS2, so that when he does IASTERISK the production

ISLASHSS2 --> IASTERISK

is stored in LTM. Next, teacher does a YP and then IMOVE2, causing the production

IASTERISKYP --> IMOVE2

to be stored. Teacher does ZZ, instead of a number action as he does in Figure VIII-8, and then YP. This puts YP into the transition context which already has SS2 in it for state 2.

Now teacher does	ISTATE	and	IMOVE2YP --> ISTATE	is stored,
then	//S2	and	ISTATESS2YP --> //S2	is stored,
then	IWRITE	and	ISTATE//S2 --> IWRITE	is stored,
then	**A	and	IWRITESS2YP --> **A	is stored,
then	IMOVE1	and	IWRITE**A --> IMOVE1	is stored,
then	MR	and	IMOVE1SS2YP --> MR	is stored.
Now teacher does another ISLASH		and	IMOVE1MR --> ISLASH	is stored.

ZZ is done so that the action that will follow it is <u>not</u> stored in a production by the slash template. The slash context must have in it "last action an identifier" to be valid. ZZ, of course, isn't an identifier action, so the slash context is invalid. When the action SS3 is done, the slash <u>rule</u> doesn't store a production with the //S2 action in the context and SS3 in the prediction. I want to be able to set up <u>any</u> state, following <u>any</u> state transition in order to teach the

transitions in a straightforward manner. Following the ZZ and SS3 the teaching continues on another cycle, and another, et cetera, until all the transitions have been taught. Then the teacher teaches the MCLS to move up and down a tape. The ZZ's after the IMOVE2 action will be replaced by number actions. First, however, the teacher must run the UTM-MCLS through each of the seven states, so that he can teach the slash productions that the ZZ's prevented being learned previously. This is straightforward. The path is through states 1, 2, 6, 3, 4, 5, 3, 4, 7, 2, 1. The teacher sets up the first state, state 1, and the symbol P, as in Figure VIII-9, to change to state 2. However, instead of the action ZZ after ISLASH, he does SS2, causing slash production

ISLASH//S2 --> SS2

to be stored. This leaves the transition context in state 2 so the teacher gives the symbol A on the next cycle, thus causing the action //S6. The teacher does SS6 after ISLASH and the slash production for state 6 is stored, and so on for the other states until all seven slash productions have been stored in LTM.

IMOVE2 ZZ MR IMOVE2 ZZ ML IMOVE2 (now the MCLS will do IMOVE2 after IMOVE2 and ML or MR) N1 MR IMOVE2 N2 MR IMOVE2 N3 MR ... N30 ML IMOVE2 N29 ML IMOVE2 N28 ... N1

Figure VIII-10

Figure VIII-10 shows the teaching sequence which teaches the productions needed for tape moving and tape adding. Tape is added when the MCLS runs out of tape part way through a computation. The only productions that are taught in Figure VIII-10 are two identifier productions and the move productions. Productions like

IMOVE2N10MR --> N11

are stored in LTM. It is then a simple matter to set up the transition context in state 2 and move along the tape to position 22 using a

sequence like that in Figure VIII-10. After that the teacher puts in the tape symbols when the MCLS does any number action for the first time. The MCLS starts off as in Figure VIII-8 and the "UTM" continues on as explained in section A.2.

Note that there are many ways to teach the productions. For example, the transitions can be taught in any order.

Notice also that tape can be added to that already "taught", part way through a computation. There was no prediction for a symbol action following the number N30. Thus teacher has the chance to do an action after N30 is performed. If teacher then starts off into the sequence in Figure VIII-10, more tape positions can be taught just as in Figure VIII-10. After N30 the identifier context is IMOVE2N30 so the teacher does an MR action, then identifier rule does IMOVE2 and the teacher does N31, et cetera. This tape adding process has been demonstrated on the UTM-MCLS.

A BASIC program for the MCLS simulation, written for a Level II, Model 1, TRS-80 microcomputer, is given in MacDonald (1980). There is also a listing of the STM and LTM of the MCLS after the transitions and 30 squares of tape have been taught.

Andreae's MCLS (Andreae, 1980a; MacDonald & Andreae, 1981) has the advantage that the tape adding is done automatically, that is without the teacher putting in any actions, once the initial teaching is finished.

CHAPTER IX

WHEN IS 'NOT' NOT 'NOT' FOR A ROBOT

It will be important for a robot to be able to do something positive in the absence of a particular condition. For example, on not recognizing something, say a person's face, a robot should do something positive on the basis of not recognizing the face, like introducing itself and saying "Hello".[1]

The problem of doing something positive in the absence of a condition will be called the "negation problem". The negation problem may be formalized like this:

> When a certain condition arises, do one thing.
>
> When that condition is absent, do another thing.
>
> There must be no special symbol or set of symbols to represent the absence of the condition.

The third sentence makes it forbidden (a) for there to be a specific symbol to signal the absence of the condition, and (b) for the system to be taught to respond with the "other thing" to every possible condition that is an absence of the "certain condition". There must be no stimulus to say that a person's face is not recognized. For example, a "Hello" problem that is similar to the negation problem has been investigated with the multiple context learning system (MCLS) PURR-PUSS (Andreae, 1975; 1977a). Briefly, in this version of the "Hello" problem, PURR-PUSS must look at a person's face, say Mr. X's, when Mr. X says "Hello". If the face is recognized, PURR-PUSS must say "Hello Mr. X". If the face is not recognized, PURR-PUSS does not predict or perform an action. Mr. X must say his name, and then say "HOW DO YOU DO". Only then does PURR-PUSS goes on to say "Hello" to Mr. X. These sequences are not a solution to the negation problem given

above. In the absence of recognition, specific stimuli---HOW DO YOU DO---signal the absence, causing PURR-PUSS to respond.[2]

In this chapter I show that an MCLS can do something positive in the absence of a condition. That is, an MCLS can "handle negation". I show that an MCLS can handle negation in three ways, each by a different MCLS. The first negation handling MCLS (NHM) was described in MacDonald (1980). The second and third negation handling MCLSs, NHM' and NHM", were described in MacDonald (1981).

Cleary (1980a; 1980b) has shown that an MFLM---a multiple memory system similar to an MCLS---cannot handle negation. Further, Cleary (1981) states that (i) the particular MCLS I used in MacDonald (1980) "conforms to the format for an MFLM" and (ii) the apparent conflict between our results arises because he accepts a sequence as having been learned only if it can be "repeated without change so long as the learning system is free to choose its own actions". Cleary goes on to show that the productions used by my 1980 negation handling MCLS, NHM, can be altered so that negation is no longer handled "correctly".

NHM' and NHM" were devised as a result of Cleary's (1981) comments. NHM" satisfies Cleary's criterion for repeatability without change. The way NHM' handles negation can be changed, but differently from the way NHM can have its handling of negation changed.

Briefly, NHM responds to the absence of a condition only if the actual condition present is a new one. NHM' responds to the absence of a condition whether the actual one is new or not. However, NHM' can be taught to respond to the presence of the particular condition with the absence response. NHM" responds to the absence of a condition whether the actual condition is new or not. Neither its presence response, nor its absence response, can be changed. NHM" satisfies Cleary's criterion for repeatability without change.

In section 1 NHM is described. In section 2 Cleary's comments are discussed, and NHM' and NHM" are described. In section 3 I suggest that NHM" is not equivalent to any MFLM and therefore is not subject to Cleary's proof that an MFLM cannot handle negation.

IX.1 NHM. NEGATION HANDLING MCLS: VERSION 1

To illustrate MCLSs that can handle negation, I consider a simple situation where the MCLS must do one thing when it receives a particular stimulus after a particular action, but another thing when it doesn't receive that stimulus after that action has occurred.

Figure IX-1 shows the actions and stimuli that will be used by NHM. Actions begin with "/" or "*", stimuli with "S".

One might imagine that the *B action here represents "open the house door", and that the following stimulus represents a person's face. SB represents seeing a familiar face, and causes the robot to "smile" and greet the person, represented by $*C_{SP}$ $*Q_{SW}$ /U. If no recognition occurs, that is if any stimulus other than SB occurs, then the robot puts out its "hand" and introduces itself, represented by $/D_{SH}$ $*H_{SI}$ /M. The SB may be thought of as representing either a particular familiar face, or the fact that the face is familiar. The actions and stimuli that occur after the *C and after the /D I chose arbitrarily, to show that a different sequence can occur after the presence of a condition, from the sequence after the absence of the condition.

I will represent the "certain situation" by <sit>, e.g open door, and the "particular condition" by <con>, e.g. recognize person. The "one thing" to do when <sit>/<con> occurs will be represented by <pres>, e.g. greet person, and the "another thing" to do when <sit>/~~<con>~~ occurs will be <abs>, e.g. introduce yourself and say "Hello". I use a crossed out <con> for the condition not being present

```
                <sit>/<con>          --->   <pres>
(a)       open door/recognize       --->   greet
            *B          SB          --->   *C  SP  *Q  SW  /U

                <sit>/<con>          --->   <abs>
(b)       open door/recognize       --->   introduce self and say "Hello"
            *B          SB          --->   /D  SH  *H  SI  /M
```

Figure IX-1. Handling negation (a) presence of a condition. (b) absence of a condition.

The Figure is explained in the text.

in order to emphasise that there can be no special symbol or set of symbols to represent not having <con>. There is no special symbol for not recognizing the person. Although face recognition is a "high level cognitive" task, it is not intended here to refer to anything more than the basic actions and stimuli of a robot. It is a convenient way for discussing the problem of a robot doing something positive when a certain condition is absent.

NHM can perform actions of two types and receives stimuli of one type from the environment. When no action is provided by the MCLS, the teacher can provide one. The fundamental thing that distinguishes the action types is that they are treated differently by the rules of the MCLS. MCLSs and their rules are explained in section 2 of chapter IV. The two types of action are *actions and /actions. *actions are predicted by a context that /actions are not predicted by, and vice-versa. So there are two separate sets of productions for * and / actions, respectively. This is useful. The teacher arranges for a /action to follow the "absence" situation and an *action to follow the "presence" situation. It is necessary to have separate sets of productions for each type of action, so that when either situation

occurs the productions that are used for the other situation are not altered. Further explanation of the need for the separate sets of productions is given at the end of this section.

All actions are members of the set [/A, /B, ... /Z, *A, *B, ... *Z], while all stimuli are members of the set [SA, SB, ... SZ]. There are three types of context, that is, three rules:-

rule 1 context template: last action
prediction template: /actions
example context-predictions: *D -> /S; /Q -> /W; *E -> /C,/D

rule 2 context template: last action and last stimulus
prediction template: *actions
example context-predictions: *B SR -> *A; /S SD -> *E,*J

rule 3 context template: last stimulus
prediction template: actions
example context-predictions: SE -> *Q; ST -> /K; SR -> *A,/D

The rules for NHM' and NHM" are the same as those for NHM. In NHM priority is the same for all three rules: there is a selection on the basis of majority evidence. Priorities for NHM' and NHM" are given later.

Three crucial teaching sequences of actions and stimuli set up the negation handling productions. Following the absence of the stimulus SB after the action *B, the sequence /D $_{SH}$ *H $_{SI}$ /M is required, as shown in Figure IX-1. Following the stimulus SB after the action *B the sequence *C $_{SP}$ *Q $_{SW}$ /U is required. The MCLS could be taught to perform any sequence after a /action has occurred in the absence situation. The MCLS could be taught to perform any sequence after an *action has occurred in the presence situation. The three crucial teaching sequences are shown below as a single sequence with %'s in place of not so important actions and stimuli:-

$\underset{SX}{\%}$ $\underset{SH}{/D}$ $\underset{SI}{*H}$ $\underset{\%}{/M}$ $\underset{\%}{\%}$ $\underset{SB}{*B}$ $\underset{SP}{*C}$ $\underset{SW}{*Q}$ $\underset{\%}{/U}$ $\underset{\%}{\%}$ $\underset{SX}{*B}$ /D *H /M
SH SI

Underlined actions were put in by the teacher. Rule 3 causes the second

Presence: - - - *B SB *C SP *Q SW /U - - -

Productions used: *B,SB -> *C *C,SP -> *Q *Q -> /U
SB -> *C SP -> *Q SW -> /U

Absence: - - - *B ~~SB~~ /D SH *H SI /M - - -

Productions used: *B -> /D /D,SH -> *H *H -> /M
SH -> *H SI -> /M

Figure IX-2 The Two situations: presence and absence of an SB after *B

/D action to occur, thus setting up the production *B -> /D, which causes /D to be performed in the absence of SB after *B. The productions formed during the sequence above are:-

rule	production	rule	production
3	SX -> /D	3	SB -> *C
2	/D,SH -> *H	2	*C,SP -> *Q
3	SH -> *H	3	SP -> *Q
1	*H -> /M	1	*Q -> /U
3	SI -> /M	3	SW -> /U
2	*B,SB -> *C	1	*B -> /D

where commas separate the action and stimulus of a context of rule 2.

This teaching sets up the sequences required for the presence and absence of an SB after an *B. Figure IX-2 shows the presence and absence sequences and the productions used. A sequence with the %'s replaced by actions and stimuli is:-

*L /D *H /M /F *B *C *Q /U /L *B /D *H /M
SX SH SI SE SG SB SP SW SK SO SX SH SI

which is a real teaching sequence. NHM was implemented both in BASIC and LISP on a TRS-80 microcomputer, and taught using the sequence immediately above. The programs are given in the appendix. The sequences that follow both situations are general. Except for the first action, any sequence of actions can follow either condition. The first action, the /D for absence or the *C for presence, may be seen as a dummy action that doesn't do anything except provide a "stepping stone" to any sequence of actions and stimuli.

I said at the start of this section that there must be separate sets of productions predicting different types of action. Rule 1 productions predict /actions. Rule 2 productions predict *actions. If, for example, rule 1 predicted *actions as well as /actions then the presence situation would cause rule 1 to predict *C as well as /D, after *B. For the absence situation to be handled properly, /D must be predicted and performed by rule 1. So *C must not be predicted by rule 1. The performance of /D must be by rule 1 because the stimulus in the contexts of rules 2 and 3 is being treated as the absence of SB. The /D must be performed by rule 1 alone. So rule 1 must predict /D. It must not predict *C.

IX.2 NEGATION HANDLING MCLS: VERSIONS 2 AND 3

Cleary (1981) has shown that my NHM responds properly to the absence of <con> only if the actual condition or stimulus has never been seen by the NHM at all.

If the actual stimulus has been seen before then when <sit>/<~~con~~> occurs the NHM is just as likely to respond to the presence of the actual stimulus as it is to the absence of <con>. So in Cleary's example-sequence the NHM may do a /Z instead of the /D, when the stimulus SB is absent but either SY, or SX, is present. Figure IX-3 shows this example-sequence. NHM predicts both /Z and /D. Its predictions are ambiguous. A corresponding stimulus or condition in the face recognition task might be the robot finding a charity collector at the door, as indicated in Figure IX-3. On recognizing the collector's badge or uniform the robot might respond by giving a donation rather than introducing itself and saying "Hello".

A slight alteration to the decision procedure of NHM forms NHM'. NHM' will respond to the absence of <con> when <sit> occurs, regardless of the actual stimulus. It will respond to <sit>/<~~con~~> with <abs>

Figure IX-3 NHM responds to the absence of SB after *B only if the actual stimulus hasn't occurred before.

(a) SY has ocurred before and predicts /Z

(b) Cleary's (1981) example sequence

	<sit>/<con>	--->	<another response>
(a)	open door/recognize badge	--->	give a donation
	*B SY	--->	/Z

(b) Cleary's example sequence follows the teaching sequence of section 1. The first /Z must be put in by the teacher. The next three /Z's are performed by NHM. In each of the three performances of /Z, /D is just as likely to be performed.

-	*X	/Z	-	*X	/Z	-	*B	/Z	-	*B	/Z
-	SY	-	-	SX	-	-	SY	-	-	SX	

Productions stored

*X -> /Z	SX -> /Z	*B -> /Z
SY -> /Z		

Productions used

*X -> /Z	*B -> /D	*B -> /Z
SX -> /D	SY -> /Z	*B -> /D
		SX -> /Z
		SX -> /D

Figure IX-4 NHM' can be taught to respond with the absence response in the presence of the condition.

```
walk down street/bump into a person  --->  introduce self & say "Hello"
   <another sit>/<another con>       --->  <abs>
      *A              SY             --->   /D
```

then

```
walk down street/recognize a person  --->  introduce self & say "Hello"
   <another sit>/<con>               --->  <abs>
      *A           SB                --->   /D
```

whether the actual condition has occurred before or not. Even if there is a collector at the door the robot will introduce itself and say "Hello". The three rules in NHM have equal weight in the selection of an action from the actions that are predicted. One very small change in the decision priorities transforms NHM into NHM'; the weight of rule 3 is reduced to one half the weight of rules 1 and 2. The changes to the programs are shown in the appendix.

Now, when <con> [or SB or recognize person] is absent, a prediction from rule 3 [recognizing a collector's badge for example] will not be as strong as the prediction, from rule 1, of <abs> [or /D or introduce self and say "Hello"].

NHM' does not satisfy Cleary's criterion for repeatability without change. In fact, as shown in Figure IX-4, the productions of NHM' can be changed so that <abs> is predicted as strongly as <pres> when <sit>/<con> occurs. The sequence $*A_{SY}$ /D . . $*A_{SB}$ /D, shown in Figure IX-4, will cause $*B_{SB}$, or <sit>/<con>, to be followed by either /D or *C, (<abs> or <pres>) with equal likelihood.

<abs> is predicted by <another sit> because of the first part of the sequence in Figure IX-4. This causes <abs> to be done in the second part of the sequence, which in turn causes <abs> to be predicted by <con>. That is, recognition predicts introduce self and say "Hello"! Now when <sit>/<con> occurs, <abs> is predicted as strongly as <pres>. NHM' ambigously predicts /D and *C. So the robot might open the door and introduce itself to someone it recognized. The correspondence between NHM' and the face recognition task is becoming a little awkward. This is at least partly because NHM' is a very simple MCLS.

An alteration can be made to NHM' so that Cleary's criterion for repeatability without change is satisfied by it. If the weight of rule 2 is increased to one and a quarter times the weight of rule 1 in NHM', then NHM" is created. The program changes are shown in the appendix. Once NHM" has learned the productions for handling negation they cannot be changed. The robot will always introduce itself to a person it doesn't recognize at the door. The robot will always greet a person it does recognize. The prediction of *C, or <pres>, by *B,SB (<sit>/<con>) in rule 2 and SB in rule 3, will have a higher weight than any prediction by SB in rule 3 and *B in rule 1. *C will always be performed after *B,SB. The prediction of /D by *B in rule 1 cannot be overcome by any prediction from any stimulus in rule 3.

The sequences given above and the program changes given in the appendix have been verified with the BASIC and LISP programs, also given in the appendix.

NHM", NHM' and NHM have the same rules and templates, but different rule priorities. The priorities of NHM' enable it to predict unambigously where NHM predicted ambiguously. The priorities of NHM" enable it to predict unambiguously where NHM' predicted ambiguously.

Thus an MCLS is capable of handling negation in three ways. In one way, by NHM, the presence of other stimuli may be responded to. In a

second way, by NHM', only a particular sequence of events can upset the handling of negation. In the third way, by NHM", Cleary's criterion for repeatability without change is satisfied.

IX.3 MFLMs AND MCLSs

I suspect that NHM" is not equivalent to any MFLM and therefore that Cleary's proof, of MFLMs' inability to handle negation, does not apply to it. Cleary's proof relies on (i) an MFLM being restricted to having a "monotonic" decision procedure and (ii) the fact that each FLM in an MFLM "predicts" every event of its type if no particular event or events can be predicted (Cleary 1980a; Cleary, 1980b).

In contrast to (i), there are no restrictions on the decision procedure of an MCLS except that it be reasonably "simple" (MacDonald & Andreae, 1981). Cleary's condition for a monotic decision procedure means that actions predicted later on in time will be subsets of actions predicted earlier in time, in the same situation. Then if the MFLM predicts only one action in a situation, at a particular time, it will from then on predict only that one action in that situation. Once the MFLM has been taught to perform a particular sequence of actions in a particular sequence of situations, it will continue to do so (Cleary, 1980a; 1980b). Now once NHM" has been taught to handle negation, its decisions are monotonic. Its predictions cannot be changed after the teaching.

In contrast to (ii), if a single context in an MCLS can't predict a particular event or events then it predicts nothing. Cleary's prediction of all events, by an FLM that is unable to predict a particular event, means that an unpredicted event may be performed ahead of a predicted one. This may happen if the predictions of different types of action are in competition for performance. By an "unpredicted" event I mean one that is not "predicted in particular",

but is predicted because no particular events can be predicted. In all three NHMs, the predictions of /actions and *actions compete. Rule 1 predicts /actions and rule 2 predicts *actions. If rules 2 and 3 predict nothing and rule 1 predicts /D then /D will be performed. This is what happens in the absence condition when the actual stimulus is a new one, as explained in section 1. However, suppose instead that each rule predicted all the actions specified by its prediction template, when there was nothing in particular to predict. Then, when rule 1 predicted /D and rules 2 and 3 couldn't predict, all the *actions would compete with /D. Rule 3 would predict all *actions and all /actions. Rule 2 would predict all *actions. So there would be two predictions of /D and two predictions of every *action. An *action might be performed instead of /D, even though there is evidence for /D in rule 1, but no evidence for an *action in any rule. So /D might not be performed in the absence situation of sections 1 and 2. For example, if this hypothetical MCLS had the priorities of NHM", then an *action would be performed, since prediction by rules 2 and 3 is stronger than prediction by rules 1 and 3. The three negation handling MCLSs rely on rules not predicting all events of their type when they can't predict particular events. They seem not to be subject to Cleary's proof.

Another important effect of FLMs predicting all events when they can't predict particular events is that there is always an action performed. One of the events is selected at random if no prediction is made. So the sequences that are like Cleary's one shown in Figure IX-3 (b), could in fact occur in an MFLM without the teacher doing anything, as Cleary notes. However for the three MCLSs NHM, NHM' and NHM" to have such sequences the teacher has to do the actions that are not done by the MCLS. The MCLSs do actions only as a result of the prediction of particular events. In this sense the changes Cleary discusses are not "unexpected and spontaneous" in NHM and NHM',

although they might be, if done by an MFLM.

Cleary has good reasons for requiring that sequences be repeatable without change in order for them to be accepted as having been learned (Cleary, 1980a). Then an algorithm that is put into an MFLM will always be present. However the sequences that a learning system has learned do not have to remain unchanged for them to be useful. The possibility of learned sequences changing is an important property for a robot, a human being or an animal to have.

It may not be appropriate for a robot to always respond to the absence of some stimulus or condition in a situation, after it has been taught to do so. The way NHM and NHM' handle negation is repeatable, but allows changes to be made. I have some difficulty, as the reader may have, deciding what the responses of a robot should be to the situations posed in this chapter. The important point this chapter makes is that an MCLS can handle negation in any of three ways.

IX.4 CONCLUSION

I have shown that an MCLS can handle the problem of negation: doing something positive when some condition is absent. There are three ways that an MCLS can handle negation. The first responds to the absence of a particular condition only if the actual condition present is a new one for the MCLS. The second responds to the absence of the particular condition regardless of whether the actual condition is new or not, but the MCLS can be taught to change its responses to the presence of the particular condition. The third always responds properly to both the presence and the absence of the particular condition. It cannot have these responses altered once they have been learned.

NOTES FOR CHAPTER IX

1. Anderson (1976) considers that

> It is essential for a system like ACT to be able to recognize when it does not know something. (p.190)

ACT, which comprises a production system and an associative memory, is Anderson's model of "human cognitive functioning".

2. Specifically, both (a) the "HOW DO YOU DO" given after the stranger says his name on pages 30 to 38 of Andreae (1975) and on pages 123 to 125 of Andreae (1977a), and (b) the sequence of null stimuli suggested for after "MICKEY MOUSE" on page 42 of Andreae (1975), cause PURR-PUSS to treat Mr. X as a stranger, looking again at his face and saying "Hello" to him. Thus PURR-PUSS's responses after scanning a face are not a solution to the negation problem. Note also that the face scanning itself is not an solution to the negation problem. In neither reference are the face "scanning" actions performed by PURR-PUSS in response to new stimuli. In Andreae (1975) those actions are not performed by PURR-PUSS, but by the teacher. In Andreae (1977a) the faces have been seen and scanned beforehand. Thus the face scanning cannot be considered a negation task. If the face scanning required PURR-PUSS to scan familiar and unfamilar faces herself, then face scanning might be a negation task. Andreae's "Hello" task demonstrates that PURR-PUSS can perform a task in which something must be said as a result of having heard something (p.120. Andreae, 1977a).

APPENDIX FOR CHAPTER IX

IX.A.1 BASIC program for NHM

Figure IX-5 gives the TRS-80 model 1 microcomputer BASIC program for NHM.

IX.A.2 BASIC program changes for NHM' and NHM"

NHM': Add line 42, "42 DEFSNGL,J". Variables L and J are now floating point variables

In line 5340, "L(LI)=1" becomes "L(LI)=0.5"
In line 5620, "J=1" becomes "J=0.5"

NHM": In line 5420, "L(I)=L(I)+1" becomes "L(I)=L(I)+1.25"; "L(LI)=1" becomes "L(LI)=1.25"

Now, NHM and NHM' may predict ambiguously, as explained in section 2. That is, they may predict, with the same priority, more than one action. NHM' gives a priority of 1.5 to both (a) predictions of just rules 1 and 3, and (b) predictions of just rules 2 and 3. NHM gives a priority of 1 to both (a) predictions of just rule 1, and (b) predictions of just rule 2. NHM gives all predictions of just a single rule a priority of 1. NHM gives all predictions of just two rules a priority of 2. WHen an ambiguous predictions is made, the MCLS still must select only one action to perform. For example, the BASIC program for NHM and NHM' chooses approximately the most recent of the highest priority actions. NHM" makes no ambiguous predictions.

IX.A.3 LISP program for NHM

Figure IX-6 (a) gives the TRS-80 model 1 LISP 3.72 program for NHM.

IX.A.4 LISP program changes for NHM' and NHM"

Figure IX-6 (b) gives the definition of the function DECIDE that is used instead of the DECIDE in Figure IX-6 (a) for NHM". NHM' is formed if the priorities in DECIDE are changed as shown below.

change	2.75	to	2.5
"	2.25	"	2
"	1.75	"	1.5
"	1.5	"	1.5
"	1.25	"	1
"	1	"	1
"	0.5	"	0.5

As stated in section A.2, the MCLS must still select one action to perform when an ambiguous prediction is made. The LISP programs test for predictions in decreasing order of priority, selecting the first prediction they find. The function DECIDE performs the selection. For example, a prediction by just rules 1 and 2 will always be selected in favour of one by just rules 2 and 3 or just rules 1 and 3, as shown by the order in which predictions are tested in DECIDE. In fact all three versions of DECIDE examine the rules' predictions in exactly the same order. Therefore all three MCLS programs will perform the same action, given any set of rule predictions. Nevertheless, it was important to make the priorities explicit in the description of NHM' and NHM". The selections of NHM and NHM' could be altered by changing DECIDE. For example, suppose that in the version of DECIDE used for NHM', the examination of rules 1 and 3 is done before rules 2 and 3. Then the program for NHM' would give different selections.

Figure IX-5 BASIC program for negation handling by NHM

Changes are given in the text for forming NHM' and NHM".

```
10 RANDOM:CLS:PRINT"TRS-80 BASIC PROGRAM TO DEMONSTRATE A ":PRINT
   "MULTIPLE CONTEXT LEARNING MACHINE THAT CAN HANDLE NEGATION":
   PRINT:PRINT:PRINT
20 PRINT"17 APRIL 1980   B A MACDONALD":PRINT:PRINT:PRINT
30 PRINT"UNIVERSITY OF CANTERBURY"
40 CLEAR 2000:DEFINTE-Z:I=0:J=0
50 DEFSTRA,S:DIMCA$(30),C2$(30),C3$(30)'LEFT HAND SIDE OF PRODUCTIONS
                                          (i.e. LTM contexts)
60 S1$="":S2$="":S3$=""'SHORT TERM MEMORY
70 DIMP1$(30,5),P2$(30,5),P3$(30,5),PRED$(30),L(30)'LONG TERM MEMORY
   AND LIST OF PREDICTIONS        there can be more than one prediction
                                        for a context
80 DEFSTRQ,P
90 I1$="*":I2$="/":I3$="S"
100 A="":NN=1:X1=-1:X2=-1:X3=-1
110 LL=1
120 Q=INKEY$:IFQ<>""THENIFQ="T"THENLL=1ELSELL=0        For testing the
                                                    the program the teacher
                                                   can override predictions
130 IFLL=1THENGOSUB1000ELSEIFA=""THENGOSUB1000'GET ACTION
140 GOSUB2000'DISPLAY ACTION
150 GOSUB6000'GET STIMULUS
160 GOSUB3000'STORE ACTION
170 GOSUB4000'UPDATE CONTEXTS
180 GOSUB5000'DO PREDICTIONS AND DECIDE ON AN ACTION
190 NN=NN+1:GOTO120

1000 'GET ACTION
1010 IFA<>""THENPRINT"PRED-"A"   ";
1020 Q="":INPUT"ACTION";Q:IFQ<>""THENA=Q
1030 IFA=""THEN1020
1040 RETURN

2000 'DISPLAY ACTION
2010 PRINTNN"   ";:PRINT"P"J"   ";:PRINTA
2020 RETURN

3000 'STORE ACTION
3010 'CONTEXT ONE     /,* PREDICT /
3020 IFLEFT$(A,1)<>I2$ORS1$=""THEN3100
3030 IFX1=-1THENK1=K1+1:C1$(K1)=S1$:P1$(K1,0)=A:GOTO3080
3040 FORI=0TO4:IFP1$(X1,I)<>ATHENIFP1$(X1,I)<>""THENNEXT:I=RND(5)-1:
     P1$(X1,I)=A:GOTO3060ELSEP1$(X1,I)=A:GOTO3070
3050 GOTO3100
3060 Q=C1$(X1):P="/ PREDICT":GOSUB10000:GOTO3100
3070 Q=C1$(X1):P="/ PREDICT":GOSUB10200:GOTO3100
3080 Q=S1$:P="/ PREDICT":GOSUB10100
3100 'CONTEXT TWO    /-S, *-S  PREDICT *
3110 IFLEFT$(A,1)<>I1$ORS2$=""THEN3200
3120 IFX2=-1THENK2=K2+1:C2$(K2)=S2$:P2$(K2,0)=A:GOTO3180
3130 FORI=0TO4:IFP2$(X2,I)<>ATHENIFP2$(X2,I)<>""THENNEXT:I=RND(5)-1:
     P2$(X2,I)=A:GOTO3160ELSEP2$(X2,I)=A:GOTO3170
3140 GOTO3200
3160 Q=C2$(X2):P="* PREDICT":GOSUB10000:GOTO3200
3170 Q=C2$(X2):P="* PREDICT":GOSUB10200:GOTO3200
3180 Q=S2$:P="* PREDICT":GOSUB10100
```

Figure IX-5 (continued)

```
3200 'CONTEXT THREE   S PREDICT *,/
3210 IFLEFT$(A,1)<>I1$ANDLEFT$(A,1)<>I2$ORS3$=""THEN3300
3220 IFX3=-1THENK3=K3+1:C3$(K3)=S3$:P3$(K3,0)=A:GOTO3280
3230 FORI=0TO4:IFP3$(X3,I)<>ATHENIFP3$(X3,I)<>""THENNEXT:I=RND(5)-1:
     P3$(X3,I)=A:GOTO3260ELSEP3$(X3,I)=A:GOTO3270
3240 GOTO3300
3260 Q=C3$(X3):P="/,* PREDICT":GOSUB10000:GOTO3300
3270 Q=C3$(X3):P="/,* PREDICT":GOSUB10200:GOTO3300
3280 Q=S3$:P="/,* PREDICT":GOSUB10100:GOTO3300
3300 RETURN

4000 'UPDATE CONTEXTS
4010 'CONTEXT ONE
4020 S1$=A
4100 'CONTEXT TWO
4110 S2$=A+S
4200 'CONTEXT THREE
4210 S3$=S
4300 RETURN

5000 'PREDICTION AND DECISION
5010 'CONTEXT ONE
5020 FORI=0TOK1
5030 IFC1$(I)=S1$THENX1=IELSENEXT:X1=-1
5100 'CONTEXT TWO
5110 FORI=0TOK2
5120 IFC2$(I)=S2$THENX2=IELSENEXT:X2=-1
5200 'CONTEXT THREE
5210 FORI=0TOK3
5220 IFC3$(I)=S3$THENX3=IELSENEXT:X3=-1
5300 'NOW FIND ACTION
5310 LI=0:FORI=0TO30:PRED$(I)="":NEXT
5320 IFX3=-1THEN5400
5330 FORJ=0TO4:IFP3$(X3,J)=""THEN5400
5340 PRED$(LI)=P3$(X3,J):L(LI)=1:LI=LI+1
5350 NEXT
5400 IFX2=-1THEN5500
5410 FORJ=0TO4:IFP2$(X2,J)=""THEN5500
5420 FORI=0TOLI-1:IFP2$(X2,J)=PRED$(I)THENL(I)=L(I)+1:GOTO5430ELSENEXT:
     PRED$(LI)=P2$(X2,J):L(LI)=1:LI=LI+1
5430 NEXT
5500 IFX1=-1THEN5600
5510 FORJ=0TO4:IFP1$(X1,J)=""THEN5600
5520 FORI=0TOLI-1:IFP1$(X1,J)=PRED$(I)THENL(I)=L(I)+1:GOTO5530ELSENEXT:
     PRED$(LI)=P1$(X1,J):L(LI)=1:LI=LI+1
5530 NEXT
5600 'FIND APPROX. MOST RECENT HIGHEST PRIORITY ACTION
5610 IFLI=0THENA="":RETURN
5620 J=1:FORI=0TOLI-1
5630 IFL(I)>=JTHENJ=L(I):A=PRED$(I)
5640 NEXT
5650 RETURN

6000 'GET STIMULUS
6010 PRINT"STIMULUS?";:
6020 S=INKEY$:IFS=""THEN6020
6030 S="S"+S:PRINTS
6040 RETURN

10000 ME$="RUN OUT OF ROOM FOR PREDICTIONS FOR THIS PRODUCTION
```

```
      RANDOM WIPE  ":PRINTI"TH  PRED WIPED    NEW PRED":GOTO10250
10100 ME$="NEW PRODUCTION":GOTO10250
10200 ME$="NEW PREDICTION"
10250 PRINT"   "ME$":-"P"    **  "Q"--->"A"    **"
10260 RETURN

40000 FORJ=0TOK1:PRINTC1$(J)"--->"P1$(J,0)      LTM print out subroutine
40010 FORI=1TO4:PRINT"  --->"P1$(J,I):NEXTI,J
40020 FORJ=0TOK2:PRINTC2$(J)"--->"P2$(J,0)
40030 FORI=1TO4:PRINT"  --->"P2$(J,I):NEXTI,J
40040 FORJ=0TOK3:PRINTC3$(J)"--->"P3$(J,0)
40050 FORI=1TO4:PRINT"  --->"P3$(J,I):NEXTI,J
40060 STOP
```

Figure IX-6 LISP program for NHM, NHM' and NHM"

The program is given in the form of keyboard input to LISP.

(a) NHM LISP program

```
(PUTPROP (QUOTE NEGATION) (QUOTE (LAMBDA NIL
   (PROG (CONTEXTMEM1 CONTEXTMEM2 CONTEXTMEM3 CONTEXT1 CONTEXT2
          CONTEXT3 PRED POINTER1 POINTER2 POINTER3) (CHR  28) (CHR  31)

          (SETQ A (QUOTE (TRS-80 LISP PROGRAM TO DEMONSTRATE A MULTIPLE
           CONTEXT LEARNING MACHINE THAT CAN HANDLE NEGATION.  1 AUGUST
           1982 B A MACDONALD UNIVERSITY OF CANTERBURY)))
          (PRINT A) (TERPRI) (SETQ A NIL)
          (SETQ CHAR1 (QUOTE *)) (SETQ CHAR2 (QUOTE /))
          (SETQ CHAR3 (QUOTE S))
    START (COND ((NULL A) (PRIN1 (QUOTE ACTION:-)) (SETQ A (READ)))
                (T (PRIN1 A) (CHR  195) (PRINT (LIST (QUOTE PRED)
                 PRED))))
          (COND ((EQ (CAR (EXPLODE A)) CHAR1) NIL)
                ((EQ (CAR (EXPLODE A)) CHAR2) NIL)
                (T (SETQ A NIL) (GO START)))
          (PRIN1 (QUOTE STIM:-))
          (SETQ STIM (IMPLODE (LIST CHAR3 (READ))))
          (STORE A)
          (SETQ CONTEXT1 A)
          (SETQ CONTEXT2 (IMPLODE (LIST A STIM)))
          (SETQ CONTEXT3 STIM)
          (SETQ A (PREDICT NIL))
          (GO START))))
(QUOTE FEXPR))

(PUTPROP (QUOTE MATCH) (QUOTE (LAMBDA (CONTEXT CONTEXTMEM)
   (COND ((NULL CONTEXT) NIL)
         ((NULL CONTEXTMEM) NIL)
         ((EQ CONTEXT (CAAR CONTEXTMEM)) CONTEXTMEM)
         (T (MATCH CONTEXT (CDR CONTEXTMEM))))))
(QUOTE EXPR))

(PUTPROP (QUOTE MEMBER) (QUOTE (LAMBDA (X Y)
   (COND ((NULL Y) NIL)
         ((EQ X (CAR Y)) T)
         (T (MEMBER X (CDR Y))))))
(QUOTE EXPR))

(PUTPROP (QUOTE UNION) (QUOTE (LAMBDA (A B)
   (COND ((NULL A) B)
         ((MEMBER (CAR A) B) (UNION (CDR A) B))
         (T (CONS (CAR A) (UNION (CDR A) B))))))
(QUOTE EXPR))

(PUTPROP (QUOTE DECIDE) (QUOTE (LAMBDA (P1 P2 P3)
   (PROG NIL (SETQ PRED  0)
             (COND ((NULL (NULL (SETQ X (INTERSECTION (INTERSECTION P1
                    P2) P3)))) (SETQ PRED  3) (GO E))
                   ((NULL (NULL (SETQ X (INTERSECTION P1 P2)))) (SETQ
                    PRED  2) (GO E))
                   ((NULL (NULL (SETQ X (INTERSECTION P2 P3)))) (SETQ
                    PRED  2) (GO E))
                   ((NULL (NULL (SETQ X (INTERSECTION P1 P3)))) (SETQ
                    PRED  2) (GO E))
                   ((NULL (NULL (SETQ X (UNION (UNION P1 P2) P3))))
```

```
                    (SETQ PRED  1) (GO E))
                  (T (RETURN NIL))) E (RETURN (CAR X)))))
(QUOTE EXPR))

(PUTPROP (QUOTE STORE) (QUOTE (LAMBDA (A)
   (PROG NIL (COND ((NULL A) RETURN)
                  ((EQ (CAR (EXPLODE A)) CHAR2) (SETQ CONTEXTMEM1
                    (STORE1 CONTEXTMEM1 CONTEXT1 A POINTER1)))
                  ((EQ (CAR (EXPLODE A)) CHAR1) (SETQ CONTEXTMEM2
                    (STORE1 CONTEXTMEM2 CONTEXT2 A POINTER2)))
                  (T (PRINT (QUOTE (ERROR: ACTION NOT * OR /))
                    (RETURN))))
             (SETQ CONTEXTMEM3 (STORE1 CONTEXTMEM3 CONTEXT3 A
               POINTER3)))))
(QUOTE EXPR))

(PUTPROP (QUOTE STORE1) (QUOTE (LAMBDA (CM C A POINT)
   (COND ((NULL POINT) (CONS (CONS C (LIST A)) CM))
         (T (COND ((MEMBER A (CDAR POINT)) NIL)
                  (T (RPLACD (CAR POINT) (CONS A (CDAR POINT)))))
            CM))))
(QUOTE EXPR))

(PUTPROP (QUOTE PREDICT) (QUOTE (LAMBDA NIL
   (PROG NIL (SETQ POINTER1 (MATCH CONTEXT1 CONTEXTMEM1))
             (SETQ PREDS1 (PREDICT1 POINTER1))
             (SETQ POINTER2 (MATCH CONTEXT2 CONTEXTMEM2))
             (SETQ PREDS2 (PREDICT1 POINTER2))
             (SETQ POINTER3 (MATCH CONTEXT3 CONTEXTMEM3))
             (SETQ PREDS3 (PREDICT1 POINTER3))
             (RETURN (DECIDE PREDS1 PREDS2 PREDS3)))))
(QUOTE EXPR))

(PUTPROP (QUOTE PREDICT1) (QUOTE (LAMBDA (POINTER)
   (COND ((NULL POINTER) NIL)
         (T (CDAR POINTER)))))
(QUOTE EXPR))

(PUTPROP (QUOTE INTERSECTION) (QUOTE (LAMBDA (A B)
   (COND ((NULL A) NIL)
         ((MEMBER (CAR A) B) (CONS (CAR A) (INTERSECTION (CDR A) B)))
         (T (INTERSECTION (CDR A) B)))))
(QUOTE EXPR))
```

(b) Definition of DECIDE for NHM". Changes are given in the text for forming NHM'.

```
(PUTPROP (QUOTE DECIDE) (QUOTE (LAMBDA (P1 P2 P3)
   (PROG NIL (SETQ PRED  0)
             (COND ((NULL (NULL (SETQ X (INTERSECTION (INTERSECTION P1
                    P2) P3)))) (SETQ PRED  2.75) (GO E))
                   ((NULL (NULL (SETQ X (INTERSECTION P1 P2)))) (SETQ
                    PRED  2.25) (GO E))
                   ((NULL (NULL (SETQ X (INTERSECTION P2 P3)))) (SETQ
                    PRED  1.75) (GO E))
                   ((NULL (NULL (SETQ X (INTERSECTION P1 P3)))) (SETQ
                    PRED  1.5) (GO E))
                   ((NULL (NULL (SETQ X P2))) (SETQ PRED  1.25) (GO E))
                   ((NULL (NULL (SETQ X P1))) (SETQ PRED  1) (GO E))
                   ((NULL (NULL (SETQ X P3))) (SETQ PRED  .5) (GO E))
                   (T (RETURN NIL)))
         E (RETURN (CAR X)))))
(QUOTE EXPR))
```

CHAPTER X

CONCLUSION

Chapter II explained that existing robots lack teachability, teachability being concerned with (a) the range of tasks that can be taught, (b) the difficulty of the tasks that can be taught, and (c) ease of teaching. Chapter II explained that existing robots lack adaptability, adaptability being concerned with (a) the range of conditions that a robot can perform a task in (b) the amount of teacher assistance required when conditions change, and (c) speed of adaption.

Chapter III dealt with the leading method, which is a natural, easy to use, widely employed method of teaching a robot a sequence of movements. The teacher moves the robot's arm through the desired sequence of movements during the training mode. The robot may repeat the movements during execution mode. However, explicit programming is needed with the popular leading method, if actions must be changed or conditional branches must be formed. In chapter III I proposed two paths of improvement which enable actions to be changed and conditional branches to be formed, using only a teacher's natural ability at leading and verbal correcting, and his knowledge of task goals. Thus the teacher need not be skilled in using programming languages, just skilled in the task to be taught.

On path 1 an on-line verbal correcting (VC) scheme is added to popular leading, enabling successive corrections to be made to action sequences. VC is stable and convergent. On path 1 a production system of corrections (PSC) is added to VC and popular leading, enabling a robot to remember and use verbally taught conditional corrections for

opposing forces.

On path 2 a goal-seeking (GS) system and VC replace the sequence of actions stored during popular leading. The teacher sets goals, leads actions and makes verbal corrections. The robot can select its own actions for achieving a sequence of goals.

Teachability is increased by the two paths because VC enables a teacher to naturally make action changes, and a PSC and GS system enable him to naturally teach conditional branches. No explicit programming is required. Adaptability is increased by the two paths because a PSC or GS system enable a robot to select its own actions for achieving either a sequence of movements, if it has a PSC, or a sequence of goals, if it has a GS system.

Path 1 enables sequences to be taught, but not goals. Path 2 enables goals to be taught, but not sequences. Chapter IV explained that a multiple context learning system (MCLS) can enable a led robot to be taught both sequences and goals. An MCLS is a multiple, extended GS system. Each extended GS system, or rule, proposes actions to perform on the basis of the rule's context, which is part of the robot's recent action-stimulus history. A decision procedure in the MCLS selects actions to perform from the combined proposals of the rules. An MCLS enables both sequences and goals to be taught by having some rules learn the sequences and others the goals. An MCLS can implement VC.

Chapter V reported demonstrations of leading with the arm-robot, a simple real arm with a simple MCLS. The teacher taught the arm-robot to lift a weight into a light beam by himself lifting the arm into the light beam. Although the weight caused the arm to sag, the arm-robot still lifted it up into the light beam.

"Leak-back", dealt with in chapter VI, is a successive overrelaxation method of assigning a value to a rule's predictions. It converges on optimal values, enabling a rule to predict the best action for reaching a goal from the current context.

The leading, guiding and teacher-programming methods may not be suitable for teaching a robot to perform eye and speech actions. Instead such actions might be preprogrammed to occur as reflex-actions, actions triggered by reflex-trigger stimuli. Actions first performed by reflex alone can be learned and performed in many situations by an MCLS. Chapter VII demonstrated the MCLS PURR-PUSS learning a reflex look-left action. PURR-PUSS does not overgeneralize its learning of reflexes.

An MCLS can learn any grammar that will fit into its very large memory. Thus Hayes's (1977) criticism of PURR-PUSS, "... So it certainly couldn't learn a grammar for any reasonably interesting subset of English" (p.10), is made in error. Since PURR-PUSS can learn any automaton (Andreae & Cleary, 1976), this should have already been clear. My demonstration of an MCLS learning to be the universal computing machine called a Turing Maching makes MCLSs' ability explicit. Showing that an MCLS can learn grammars in this way takes no account of the need for an MCLS to learn, survive or interact in its environment.

Chapter IX showed that an MCLS can handle the problem of negation, that of doing something positive in the absence of a certain condition.

This thesis advances the design of teachable adaptable robots by (a) proposing improvements to leading that enable taught actions to be changed, conditional branches to be formed, and goals to be set, but that do not require the teacher to be an experienced programmer, (b) extending the improvements to the MCLS, enabling sequences of both goals and movements to be taught, and (c) showing fundamental abilities of MCLSs.

REFERENCES

Albus, J.S. (1975) Data Storage in the Cerebellar Model Articulation Controller (CMAC). Trans. ASME J. Dynamic Syst. Measurement and Contr., September : 228-33

Albus, J.S. (1979) A Model of the Brain for Robot Control. Part 2: A Neurological Model. Byte, July: 54-95.

Allan, R. (1979) Busy robots spur productivity. IEEE Spectrum, 16(9): 31-6.

Ambler, A.P., Barrow, H.G., Brown, C.M., Burstall, R.M. and Popplestone, R.J. (1975) A Versatile System for Computer-Controlled Assembly, Artificial Intelligence, 6: 129-56.

Ambler, A.P., Popplestone, R.J. and Kempf, K.G. (1982) An Experiment with the Offline Programming of Robots. Proc. 12th Int. Symp. on Industrial Robots; 6th Int. Conf. on Industrial Robot Technology. Paris. France. June. p.491-504.

Anderson, J.R. (1976) Language, Memory, and Thought. N.J., Lawrence Erlbaum Assocs. 546p.

Andreae, J.H. (1972-83) Man-Machine Studies Progress Reports UC-DSE/1-22. editor. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand.

Andreae, J.H. (1973-83) Man-Machine Studies Progress Reports UC-DSE/2-22. editor. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand.

Andreae, J.H. (1973) Work on a "River" Program : PUSS. Man-Machine Studies Progress Report UC-DSE/3. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.6-43.

Andreae, J.H. (1974) PURR-PUSS : Purposeful Unprimed Rewardable Robot. Man-Machine Studies Progress Report UC-DSE/4. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.100-50.

Andreae, J.H. (1975) PUSS Holography, Brain Models & Other Amewsments. Man-Machine Studies Progress Report UC-DSE/7. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.13-48.

Andreae, J.H. (1977a) Thinking with the Teachable Machine. London, Academic Press. 178p.

Andreae, J.H. (1977b) Response to Pat Hayes. AISB Quarterly, 26: 14-5.

Andreae, J.H. (1977c) Pushthru Computing and Context Control. Man-Machine Studies Progress Report UC-DSE/12. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.15-31.

Andreae (1977d). See Andreae, P.M.

Andreae, J.H. (1978) Computation and Thought. Man-Machine Studies Progress Report UC-DSE/13. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.18-45.

Andreae, J.H. (1979a) A Machine to Think Like Man. Man-Machine Studies Progress Report UC-DSE/14. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.9-41.

Andreae, J.H. (1979b) Muscles for Thought. Man-Machine Studies Progress Report UC-DSE/15. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.19-42.

Andreae, J.H. (1979c) Rejoinder to Professor L.K. Schubert's Review of John H. Andreae, Thinking with the Teachable Machine. Alberta J. Educ. Res., (3) September : 204-5.

Andreae, J.H. (1980a) Bicameral Mind. Man-Machine Studies Progress Report UC-DSE/16. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.39-73.

Andreae, J.H. (1980b) A New PURR-PUSS. Man-Machine Studies Progress Report UC-DSE/18. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.5-28.

Andreae, J.H. (1981) A Multiple Context Digital Computer. Man-Machine Studies Progress Report UC-DSE/19. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.74-92.

Andreae, J.H. (1982a) A Robot Mentality. Man-Machine Studies Progress Report UC-DSE/20. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.69-128.

Andreae, J.H. (1982b) What Does PURR-PUSS Need to Recall a Fact? Man-Machine Studies Progress Report UC-DSE/21. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.64-79.

Andreae, J.H. and Andreae, P.M. (1979) Machine Learning with a Multiple Context. Proc. 9th Int.Conf.on Cybernetics and Society. Denver. October. p.734-9.

Andreae, J.H. and Cashin, P.M. (1969) A Learning Machine with Monologue. Int.J.Man-Machine Studies, 1:1-20.

Andreae, J.H. and Cleary, J.G. (1976) A New Mechanism for a Brain. Int.J.Man-Machine Studies, 8(1): 89-119

Andreae, J.H. and MacDonald, B.A. (1981) Learning with Finite Automata AISB Quarterly, 40-41: 29.

Andreae, P.M. (1977d) A Teachable Machine in the Real World. Man-Machine Studies Progress Report UC-DSE/11. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.53-66.

Andreae, P.M. and Andreae, J.H. (1978) A Teachable Machine in the Real World. Int.J.Man-Machine Studies, 10: 301-12.

Anokhin, P.K. (1974) Biology and Neurophysiology of the Conditioned Reflex and its Role in Adaptive Behaviour. Oxford, Pergamon Press. 574p.

Arbib, M.A. (1972) The Metaphorical Brain. N.Y., Wiley. 243p.

Arbib, M.A. (1979) Perceptual Structures and Distributed Motor Control. COINS Technical Report 79-11, June. Preprint of a chapter for Motor Control, vol.III of Handbook of Physiology, edited by V.B.Brooks, 1980.

Astrom, K.J., Borisson, U., Ljung, L. and Wittenmark, B. (1977) Theory and Applications of Self-Tuning Regulators. Automatica, 13: 457-76.

Astrop, A. (1982) GM enters the robot field in a big way. Machinery and production engineering, 140(3598): 157-9.

Ayres, R. and Miller, S. (1982) Industrial Robots on the Line. Technology Review, May/June : 35-44.

Bahill, A.T. and Stark. L. (1979) The Trajectories of Saccadic Eye Movements. Sci. American, January: 85-93.

Beecher, R.C. (1979) PUMA: Programmable Universal Machine for Assembly. In Dodd and Rossol. p.141-9.

Bellman, R. (1961) Adaptive Control Processes: A Guided Tour. N.J., Princeton University Press. 255p.

Benati, M., Gaglio, S., Morasso, P., Tagliasco, V. and Zaccaria, R. (1980) Anthropomorphic Robotics. I.Representing Mechanical Complexity. II.Analysis of Manipulator Dynamics and the Output Motor Impedance. Biol.Cybern., 38: 125-40 and 141-50.

Bertino, M., Furxhi, M.G. and Gola, M. (1980) Microcomputer Control for a 5 Axis Manipulator with Cartesian Path Control. Proc. 10th Int. Conf. on Cybernetics and Society. Cambridge. Mass. October. p.1078-84.

Bertsekas, D.P. (1976) Dynamic Programming and Stochastic Control. N.Y., Academic Press. 397p.

Berwick, R.C. and Weinberg, A.S. (1982) Parsing Efficiency, Computational Complexity, and the Evaluation of Grammatical Theories, Linguistic Inquiry, 13(2) : 165-91.

Birk, J.R., Kelley, R.B. and Martins, H.A.S. (1981) An Orienting Robot for Feeding Workpieces Stored in Bins. IEEE Trans.Syst.Man and Cybern., SMC-11(2): 151-60.

Bizzi, E., Dev, P., Morasso, P. and Polit, A. (1978) Effect of Load Disturbances During Centrally Initiated Movements. J. Neurophys., 41(3): 542-56.

Blakeslee, T.R. (1980) The Right Brain. A new understanding of the unconcious mind and its creative powers. London, Macmillan. 437p.

Bonner, S. and Shin, K.G. (1982) A Comparative Study of Robot Languages. Computer, 15(12) : 82-96.

Brooks, R.A. (1982) Symbolic Error Analysis and Robot Planning. AI Memo No. 685. MIT AI Lab. September.

Brown, R. (1973) A First Language : The Early Stages. London, George Allen & Unwin. 437p.

Cashin, P.M. (1970) Computing Procedures for a Learning Machine. Christchurch, University of Canterbury. Thesis: Ph.D.: Electrical Engineering. 216p.

Chien, R.T. and Weissman, S. (1973) Planning and Execution in Incompletely Specified Environments. Proc.Int.J.Conf.on AI. Stanford. Cal. August. p.169-74.

Chomsky, N. (1959) On Certain Formal Properties of Grammars. Information and Control, 2(2) : 137-67.

Chomsky, N. (1963) Formal Properties of Grammars. In Luce et al., p.323-418.

Chomsky, N. (1965) Aspects of the Theory of Syntax. Mass., MIT Press. 251p.

Chomsky, N. and Miller, G.A. (1963) Introduction to the Formal analysis of Natural Language. In Luce et al. p.269-321.

Cleary, J.G. (1980a) An Associative and Impressible Computer. Christchurch, University of Canterbury. Thesis : Ph.D : Electrical Engineering. 319p.

Cleary, J.G. (1980b) An Associative and Impressible Computer. Man-Machine Studies Progress Report UC-DSE/17. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.5-74.

Cleary J.G. (1981) A Comment on Negation. Man-Machine Studies Progress Report UC-DSE/19. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.5-6.

Clocksin, W.F., Barrat, J.W., Davey, P.G., Morgan, C.G. and Vidler, A.R. (1982) Visually Guided Robot Arc-Welding of Thin Sheet Steel Pressings. Proc. 12th Int. Symp. on Industrial Robots; 6th Int. Conf. on Industrial Robot Technology. Paris. France. June. p.225-30.

Compressed Air Magazine (1981) Industrial Robots-The Next Generation. August: 8-14.

Cronshaw, A.J. (1982) Automatic Chocolate Decoration by Robot Vision. Proc. 12th Int. Symp. on Industrial Robots; 6th Int. Conf. on Industrial Robot Technology. Paris. France. June. p.249-57.

Crossman E.R.F.W. (1953) Entropy and Choice Time : The Effect of Frequency Unbalance on Choice - Response. The Quarterly J. of Experimental Psychology V(2) : 41-51

Crossman, E.R.F.W. and Goodeve, P.J. (1963) Feedback Control of Hand-Movement and Fitt's Law. Inst. of Experimental Psychology. Oxford Univ.

Cunningham, C.S. (1979) Robot Flexibility Through Software. 9th Int. Symp. on Industrial Robots. Washington D.C. p.297-307.

Deutsch, J.A. (1960) The Structural Basis of Behaviour. London. Cambridge University Press. 186p.

Didday, R.L. and Arbib, M.A. (1975) Eye Movements and Visual Perception : A "Two Visual System" Model. Int. J. Man-Machine Studies, 7(4) : 547-69.

Di Stephano, J.J., Stubberud, A.R. and Williams, I.J. (1967) Feedback and Control Systems. Schaum's outline series. N.Y., McGraw-Hill. 371p.

Dobrotin, B. and Lewis, R. (1977) A Practical Manipulator System. Proc.Int.J.Conf.on AI. Cambridge, Mass. August. 723-32.

Dodd, G.G. and Rossol, L. (1979) Computer Vision and Sensor-Based Robots. Symposium held at GM laboratories. editors. N.Y., Plenum. 363p.

Doddington, G.R. and Schalk, T.B. (1981) Speech recognition: turning theory to practice. IEEE Spectrum, 18(9): 26-32.

Dubowsky, S. and DesForges, D.T. (1979) The Application of Model-Referenced Adaptive Control to Robotic Manipulators. Proc. J. Auto. Contr. Conf. Denver. June. p.507-13.

Duysens, J. and Loeb, G. (1978) Commentary in Roland. p.149-50.

Dvoretzky, A. (1956) On Stochastic Approximation. Proc. 3rd Berkeley Symp. on Mathematical Statistics & Probability, 1:39-55.

Dyhre-Poulsen, P. (1978) Commentary in Roland. p.150

Ejiri, M., Uno, T., Yoda, H., Goto, T. and Takeyasu, K. (1972) A Prototype Intelligent Robot that Assembles Objects from Plan Drawings. IEEE Trans. Computers, C-21(2): 161-70

Elgerd, O.I. (1967) Control Systems Theory. Tokyo, McGraw-Hill. 562p.

Engelberger, J.F. (1979) Stand-alone vs. Distributed Robotics. In Dodd and Rossol. p.263-70.

Fel'dbaum, A.A. (1960) Dual Control Theory, I-IV. Reprinted in Oldenburger (1966). p.458-96.

Fender, D.H. (1964a) Control Mechanisms of the eye. Sci. American, July : 24-33.

Fender, D.H. (1964b) The Eye-Movement Control System : Evolution of a Model. In Reiss. p.306-24.

Finkel, R., Taylor, R., Bolles, R., Paul, R. and Feldman, J. (1975) An Overview of AL, A Programming System for Automation. Proc.Int.J.Conf.on AI. Tbilisi. Georgia. p.758-65.

Flanagan, J.L. (1981) Synthesis and recognition of speech: Teaching Computers to listen. Bell Labs. Record, May/June: 146-51.

Flanagan, J.L. (1982) Talking with Computers: Synthesis and Recognition of Speech by Machines. IEEE Trans. Biomedical Engineering, BME-29(4): 223-32.

Freedy, A., Hull, F.C., Lucaccini, L.F. and Lyman, J. (1971) A Computer-Based Learning System for Remote Manipulator Control. IEEE Trans. Syst. Man and Cybern, SMC-1(4) : 356-63.

Freund, E. (1982) Fast Nonlinear Control with Arbitrary Pole - Placement for Industrial Robots and Manipulators. Int. J. of Robotics Research, 1(1) : 65-78.

Fu, K.S. (1971) Learning Control Systems and Intelligent Control Systems: An Intersection of Artificial Intelligence and Automatic Control. IEEE Trans. Auto.Contr., AC-16(1): 70-2

Gaines, B.R. (1969) Adaptive Control Theory (The Structural and Behavioural Properties of Adaptive Controllers). In Meetham, A.R. editor. Encyclopaedia of Linguistics, Information and Control. Oxford, Pergamon. p.1-9.

Gittins, J.C. (1979) Bandit Processes and Dynamic Allocation Indices. J.Royal Statis.Soc. Series B, 41(2): 148-77.

Goksel, K., Knowles, K.A. and Kittis, L. (1976) A Microprocessor Controlled Industrial Manipulator Programmable-By-Teaching. IECI'76 Conf. Proc. "Industrial Applications" of Microprocessors, process measurement and failure mode synthesis. Philadel. Pennsylv. March. p.78-80.

Gowitzke, B.A. and Milner, M. (1980) Understanding the Scientific Bases of Human Movement. 2nd edition. Baltimore, Williams and Wilkins. 376p.

Granit, R. (1978) Commentary in Roland. p.152.

Grossberg S. (1982) Studies of Mind and Brain : Neural Principles in Learning, Perception, Development, Cognition and Motor Control. Dordrecht, D. Reidel. 662p.

Grossman, D.D. and Taylor, R.H. (1978) Interactive Generation of Object Models with a Manipulator. IEEE Trans.Syst.Man and Cybern., SMC-8(9): 667-79.

Hanson, A.R. and Riseman, E.M. (1978) Computer Vision Systems. N.Y., Academic Press. 390p.

Hartley, R., Thomas, L.C. and White, D.J. (1980). Recent Developments in Markov Decision Processes. editors. London, Academic press. 334p.

Hasegawa, T. (1982) An Interactive System for Modelling and Monitoring a Manipulation Environment. IEEE Trans. Syst. Man and Cybern., SMC-12(3) : 250-8.

Hasegawa, T. and Inoue, H. (1979) Modelling and Monitoring a Manipulation Environment. Proc.Int.J.Conf. on AI. Tokyo. August. p.369-71.

Haugan, K.M. (1974) Spray Painting Robots : Advanced Paint Shop Automation. The Industrial Robot, December: 270-2.

Haugan, K.M. and Jarvis, D.E. (1974) Electronically Controlled Manipulator for Spray Gun Applications. The Industrial Robot, March: 119-21.

Hayes, P. (1977) Small Brains and Large Minds. AISB Quarterly 26 : 10-3.

Hebb, D.O. (1958) A textbook of Psychology. Phil., W.B. Saunders. 276p.

Hohn, R.R. (1979) Robot Control Systems and Applications. Proc. J. Auto. Contr. Conf, Denver. Col. June. p.750-3.

Holland, S.W., Rossol, L. and Ward, M.R. (1979) CONSIGHT-1: A Vision controlled Robot System for transferring parts from Belt Conveyors. In Dodd and Rossol. p.81-97.

Hollerbach, J.M. (1980) A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity. IEEE Trans.Syst.Man and Cybern., SMC-10(11): 730-6.

Hollerbach, J.M. (1982) Computers, Brains, and the Control of Movement. AI Memo No. 686 MIT AI Lab. June. 12p.

Hopcroft, J.E. and Ullman, J.D. (1979) Introduction to Automata Theory, Languages, and Computation. Mass., Addison-Wesley. 418p.

Horn, B.K.P. (1979) Kinematics, Statics and Dynamics of Two-Dimensional Manipulators. In Winston and Brown (1979b). p.275-308.

Howard, R.A. (1960) Dynamic Programming and Markov Processes. Mass., MIT Press. 136p.

Howard, R.A. (1971) Dynamic Probabilistic Systems. Vols.I & II. N.Y., Wiley. 576 : 577 - 1108p.

Hyman, R. (1953) Stimulus Information as a Determinant of Reaction Time. J. of Experimental Psychology, 45(3) : 188-96.

Iannone, A.M. (1978) Commentary in Roland. p.153.

Industrial Robot, The (1982) New products from Trallfa. June. p.78.

Inoue, H. (1979) Force Feedback in Precise Assembly Tasks. In Winston and Brown (1979b). p.223-41.

Jarvis, R.A. (1982) A Computer Vision and Robotics Laboratory. Computer, 15(6) : 8-24.

Kain, R.Y. (1972) Automata Theory : Machines and Languages. N.Y., McGraw-Hill. 301p.

Kelly, R.B., Birk, J.R., Martins, H.A.S. and Tella, R. (1982) A Robot System which Acquires Cylindrical Workpieces from Bins. IEEE Trans. Syst. Man and Cybern., SMC-12(2) : 204-13.

Kelso, J.A.S. (1978) Commentary in Roland. p.153-4.

Kemeny, J.G. & Snell, J.L. (1960) Finite Markov Chains. N.J., D. Van Nostrand. 210p.

Kirchman, B., Kopecky, P. and Zdrahal, Z. (1977) Goalem from Prague. Proc.Int.J.Conf.on AI. Cambridge, Mass. August. p.771.

Kreyszig, E. (1972) Advanced Engineering Mathematics. 3rd ed. N.Y., Wiley. 866p.

Kruger, R.P. and Thompson, W.B. (1981) A Technical and Economic Assessment of Computer Vision for Industrial Inspection and Robotic Assembly. IEEE Proc., 69(12); 1524-38.

Kuiper, K. (1980a) PURR-PUSSful Language Learning. Man-Machine Studies Progress Report UC-DSE/16. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.5-10.

Kuiper, K. (1980b) Minimal Grammar and Stock Auctioneering. Man-Machine Studies Progress Report UC-DSE/18. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.29-50.

Kumar, P.R. and Seidman, T.I. (1981) On the Optimal Solution of the One-Armed Bandit Adaptive Control Problem. IEEE Trans. Auto. Contr., AC-26(5) : 1176-84

Kuo, B.C. (1962) Automatic Control systems. London, Prentice-Hall. 504p.

Langill, A.W. (1965) Automatic Control Systems Engineering. Vol.II. Advanced Control Systems Engineering. N.J., Prentice-Hall. 390p.

Lee, C.S.G. (1982) Robot Arm Kinematics, Dynamics, and Control. Computer, 15(12) : 62-80.

Lerner, E.J. (1980) Computers that see. IEEE Spectrum, 17(10): 28-33.

Lippold, O. (1973) The Origin of the Alpha Rhythm. Edinburgh, Churchill-Livingston. 267p.

Llinas, R.R. (1975) The Cortex of the Cerebellum. Sci.American, January: 56-71.

Lozano-Perez, T. (1979) A Language for Automatic Mechanical Assembly. In Winston and Brown (1979b). p.245-71.

Lozano-Perez, T. (1983) Robot Programming. Invited paper IEEE Proc. special issue on Robotics, 71(7) : 821-41.

Luce, R.D., Bush, R.R. and Galanter, E. (1963) Handbook of Mathematical Psychology. editors. Volume 2. N.Y., Wiley. 606p.

Luh, J.S.Y. and Lin, C.S. (1982) Scheduling of Parallel Computation for a Computer-Controlled Mechanical Manipulator. IEEE Trans. Syst.Man and Cybern., SMC-12(2): 214-34.

Luh, J.S.Y., Walker, M.W. and Paul, R.P.C.(1980) Resolved-Acceleration Control of Mechanical Manipulators. IEEE Trans.Auto.Contr., AC-25(3): 468-74.

MacDonald, B.A. (1979) Reflexes for Learning and Robot Vision. Man-Machine Studies Progress Report UC-DSE/15. Dept. of Electrical and Electronic Engineering, University of Canterbury, Christchurch. New Zealand. p.43-75.

MacDonald, B.A. (1980) Turing Machine Power for a Multiple Context Learning System. Man-Machine Studies Progress Report UC-DSE/16. Dept. of Electrical and Electronic Engineering, University of Canterbury, Christchurch. New Zealand p.11-38.

MacDonald, B.A. (1981) A Robot Learns by being Led through Movements. Man-Machine Studies progress Report UC-DSE/19. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.7-73.

MacDonald, B.A. (1982a) Leading Teaches Robot Any Movement Task. Man-Machine studies Progress Report UC-DSE/20. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.5-68.

MacDonald, B.A. (1982b) Enhancing the Leading Method. Man-Machine Studies Progress Report UC-DSE/21. Dept. of Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p.5-46.

MacDonald, B.A. and Andreae, J.H. (1981) The Competence of a Multiple Context Learning System. Int.J.Gen.Systems, 7: 123-37.

McGhee, R.B. (1979) Future Prospects for Sensor-Based Robots. In Dodd and Rossol. p.323-34.

McGuigan, F.J. & Lumsden, D.B. (1973) Contemporary Approaches to Conditioning and Learning. editors. Washington, D.C., V.H. Winston & Sons. 321p.

McLaughlin, J.R. (1982) TRIG : An Interactive Robotic Teach System. MIT AI Lab. Working Paper 234. June. 53p.

Marteniuk, R.G. (1976) Information Processing in Motor Skills. N.Y., Holt, Rhinehart and Winston. 244p.

Mason, M.T. (1981) Compliance and Force Control for Computer Controlled Manipulators. IEEE Trans. Syst. Man and Cyber. SMC-11(6) : 418-32.

Michie, D. (1979) Machine Models of Perceptual and Intellectual Skills. Chapter 4 in Scientific Models and Man, edited by H.Harris, Oxford, Oxford Univ. Press. p.56-78.

Miller, G.A. and Chomsky, N. (1963) Finitary Models of Language Users. In Luce et al. p. 419-91.

Miller, J.A. (1977) Autonomous Guidance and Control of a Roving Robot. Proc.Int.J.Conf.on AI. Cambridge, Mass. August. p.759-60.

Minsky, M.L. (1967) Computation : Finite and Infinite Machines. N.J., Prentice-Hall. 317p.

Mishkin, E. and Braun, L. (1961) Adaptive Control Systems. editors. N.Y., McGraw-Hill. 533p.

Morris, H.M. (1982) Where Do Robots Fit in Industrial Control? Control Engineering, 29(3): 58-64.

Nagel, R.N. (1983) Robots : not yet smart enough. IEEE Spectrum, 20(5) : 78-83.

Nelson, R.J. (1978) The Competence - Performance Distinction in Mental Philosophy. Synthese 39 : 337-81.

Nevins, J.L. and Whitney, D.E. (1979) Robot Assembly Research and its Future Applications. In Dodd and Rossol. p.275-321

Newell, A. (1982) The Knowledge Level. Artificial Intelligence, 18 : 87-127.

Nilsson, N.J. (1980) Principles of Artificial Intelligence. Palo Alto, Tioga. 476p.

Nishimoto, Katsushi, Uchiyama, Takashi and Akita, Tadashi (1983) Development of a precision robot. Automation and Control, 13(4) : 21-4.

Nitzan, D. (1979) Flexible Automation Program at SRI. Proc. J. Auto. Contr. Conf. Denver. June. p.754-9.

Noton, D. and Stark, L. (1971) Eye Movement and Visual Perception. Sci. American, June : 34-43.

Oim, H. (1981) Language, Meaning and Human Knowledge. Nordic J. of Linguistics, 4 : 67-90.

Oldenburger, R. (1966) Optimal and Self-Optimizing Control. editor. Mass., MIT Press. 500p.

Palfi, A.F. (1976) Controlling a Robot by PURR-PUSS, A Teachable Machine. Man-Machine Studies Progress Report UC-DSE/9. Dept. Electrical and Electronic Engineering, University of Canterbury. Christchurch New Zealand. p. 5-28.

Palfi, A.F. (1977a) A Real Time "Body" and "World" for PURR-PUSS. Man-Machine Studies Progress Report UC-DSE/10. Dept. Electrical and Electronic Engineering, University of Canterbury. Christchurch New Zealand. p. 6-24.

Palfi, A.F. (1977b) Timing Mechanisms for PURR-PUSS. Man-Machine Studies Progress Report UC-DSE/12. Dept. Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p. 49-67.

Park, W.T. and Burnett, D.J. (1979) An interactive Incremental Compiler for more Produtive Programming of Computer Controlled Industrial Robots and Flexible Automation Systems. Proc. 9th Int. Symp. on Industrial Robots. Washington D.C., March p.281-95.

Paul, R. (1979) Robots, Models and Automation. Computer, 12(7) : 19-27.

Paul, R. (1981) Robot Manipulators : Mathematics, Programming, and Control. The Computer Control of Robot Manipulators. Mass. MIT Press. 279p.

Pegg, S. (1983a) Multiple Processors and a Large Expandable Memory for a Learning System. Master of Engineering Report, University of Canterbury. 109p.

Pegg, S. (1983b) Implementing Leakback in a PURR-PUSS with a Large Memory. Man-Machine Studies Progress Report UC-DSE/22. Dept. Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p. 32-6.

Pew, R.W. (1974) Levels of Analysis in Motor Control. Brain Research, 71: 393-400.

Pipes, L.A. and Harvill, L.R. (1970) Applied Mathematics for Engineers and Physicists. 3rd edition. Tokyo, McGraw-Hill. 1015p.

Popplestone, R.J., Ambler, A.P. and Bellos, I.M. (1980) An Interpreter for a Language Describing Assemblies. Artificial Intelligence, 14 : 79-107.

Pribram, K.J. (1969) On the Biology of Learning. editor. N.Y., Harcourt, Brace & World. 225p.

Pribram, K.H. (1978) Commentary in Roland. p.158-9.

Production Engineer, The (1982) Painting Robots Halve Cycle Times for Subcontractor. May : 50-1.

Rabischong, P., Peruchon, E. and Pech, J. (1977) Is Man Still the Best Robot? Proc. 7th Int. Symp. on Industrial Robots. Tokyo. October. p.49-57.

Rachlin, H. (1976) Behaviour and Learning. San Francisco, Freeman. 613p.

Raibert, M.H. (1978) A Model for Sensorimotor Control and Learning. Biol.Cybern., 29: 29-36.

Raibert, M.H. and Horn, B.K.P. (1978) Manipulator Control using the Configuration Space method. The Industrial Robot, June: 69-73.

Raphael, B. (1976) The Thinking Computer. San Francisco, Freeman. 322p.

Reddy, D.R. and Hon, R.W. (1979) Computer Architectures for Vision. In Dodd and Rossol. p.169-85.

Reetz, D. (1973) Solution of a Markovian Decision Problem by Successive Overrelaxation. Zeitschrift für Operations Research. 17 : 29-32.

Reiss, R.F. (1964) Neural Theory and Modelling. editor. Proc. 1962 Ojai Symp. Cal., Stanford Univ. Press. 427p.

Rescorla, R.A. (1973) Second-Order Conditioning: Implications for Theories of Learning. In McGuigan & Lumsden. p.127-50.

Roland, P.E. (1978) Sensory Feedback to the Cerebral Cortex during Voluntary Movement in Man. The Behavioural and Brain Sciences, 1: 129-17

Rosen, C.A. (1979) Machine Vision and Robotics: Industrial Requirements. In Dodd and Rossol. p.3-20.

Rushby, R.J. Brander, W.D.S., Blythe, C.G., Byron, N.C. and Nowland, P.J. (1975) CAESAR in ROME. Man-Machine Studies Progress Report UC-DSE/7. Dept. Electrical and Electronic Engineering, University of Canterbury. Christchurch. New Zealand. p. 4-12.

Sachs, J. and Leifer, L. (1979) Voice Command of a Six-Degree-of Freedom Manipulator. Proc. J. Auto. Contr. Conf. Denver. June : 783-89.

Sakitt, B. (1980) A Spring Model and Equivalent Neural Network for Arm Posture Control. Biol.Cybern., 37: 227-34.

Salman, M. and d'Auria, A. (1979) Programmable Assembly System. In Dodd and Rossol. p.153-63.

Saltz, E. (1973) Higher Mental Processes as the Bases for the Laws of Conditioning. In McGuigan & Lumsden. p.21-47.

Schubert, L.K. (1978) Review of Thinking with the Teachable Machine, by J.H. Andreae. Alberta J. of Educ. Research, xxiv(4) : 291-2

Seltzer, D.S. (1979) Use of Sensory Information for Improved Robot Learning. Dearborn, Michigan, Soc. Manufacturing Engineers (SME Technical Paper; MS 79-799. 11p.

Selverston, A.J. (1980) Are central pattern generators understandable? The Behavioural and Brain Sciences, 3: 535-72.

Shebilske, W.L. (1978) Commentary in Roland. p.160-1.

Sheridan, T.B., Brooks, T.L., Takahashi, M. and Ranadive, V. (1979) How to Talk to a Robot. Proc.Int.Conf.on Cybern. and Society Denver. October. p.33-5.

Sheridan, T.B. and Ferrell, W.R. (1981) Man-Machine Systems. Information, Control, and Decision Models of Human Performance. 2nd ed. Cambridge, Mass. MIT Press. 707p.

Shirai, Y. (1979a) A panel on Industrial Applications of Robotics. Chairman. Proc.Int.J.Conf.on AI. Tokyo. August. p.1107.

Shirai, Y. (1979b) Three-dimensional Computer Vision. In Dodd and Rossol. p.187-205.

Simons, G.L. (1980) Robots in Industry. Manchester, National Computing Centre. 216p.

Skinner, B.F. (1938) The behaviour of organisms. N.Y., Appleton-Century-Crofts. 457p.

Spiegel, M.R. (1968) Mathematical Handbook of Formulas and Tables. N.Y., McGraw-Hill. 271p.

Stark, L. (1968) Neurological Control systems. Studies in Bioengineering. N.Y., Plenum. 428p.

Stein, J. (1978) Commentary in Roland. p.162-3.

Stein, R.B. and Oguztoreli, M.N. (1981) The Role of γ-Motoneurons in Mammalian Reflex Systems. Biol.Cybern., 39: 171-9.

Sutton, R.S. and Barto, A.G. (1981) Toward a Modern Theory of Adaptive Networks : Expectation and Prediction. Psych. Review, 88(2) : 135-70.

Takahashi, M. and Kohno, M. (1982) An Assembly Robot System with Twin Arms and Vision. Proc. 12th Int. Symp. on Industrial Robots; 6th Int. Conf. on Industrial Robot Technology. Paris. France. June. p.111-20.

Takase, K. (1979) Skill of Intelligent Robot. Proc.Int.J.Conf.on AI. Tokyo. p.1095-100.

Takase, K., Paul, R.P. and Berg, E.J. (1981) A Structured Approach to Robot Programming and Teaching. IEEE Trans.Syst.Man and Cybern., SMC-11(4): 274-89.

Takegaki, M. and Arimoto, S. (1981) An Adaptive Trajectory Control of Manipulators. Int. J. Control, 34(2) : 219-30.

Taylor, R.H. and Grossman, D.D. (1983) An Integrated Robot System Architecture. Invited paper IEEE Proc. special issue on Robotics. 71(7) : 842-56.

Taylor, R.H., Summers, P.D. and Meyer, J.M. (1982) AML : A Manufacturing Language. Int. J. of Robotics Research, 1(3) : 19-41.

Tenenbaum, J.M., Barrow, H.G. and Bolles, R.C. (1979) Prospects for Industrial Vision. In Dodd and Rossol. p.239-56.

Tolkmitt, F.J. (1976) A Computer Simulation Model of the Afferent Part of the Visual Foveation System. Max-Planck Institut für Biophysikalishe. Chemie, Göttingen. 17p.

Treer, K.R. (1979) Automated Assembly. Dearborn, Michigan, Soc. Manufacturing Engineers. 459p.

Turing, A.M. (1936) On Computable Numbers, with an Application to the Entscheidungsproblem. Proc. London Math. Soc. 42(3) : 230-65.

Uno, T., Ikeda, S., Ueda, H. and Ejiri, M. (1979) An Industrial Eye that recognizes hole positions in a water pump testing process. In Dodd and Rossol. p.101-14.

Vaccari, J.A. (1982) Robots that paint can create jobs. American Machinist, January: 131-4.

van Der Wal, J. (1981) Stochastic Dynamic Programming. Successive approximations and nearly optimal strategies for Markov decision processes and Markov Games. Mathematical Centre Tracts 139. Mathematical Centre, 413 Kruislaan, Amsterdam. 251p.

van Dijk, J.H.M. (1978a) Simulation of Human Arm Movements Controlled by Peripheral Feedback. Biol.Cybern., 29 : 175-86.

van Dijk, J.H.M. (1978b) On the Interaction between the Central Nervous System and the Peripheral Motor System. Biol.Cybern., 30: 195-208.

Varga, R.S. (1962) Matrix Iterative Analysis. N.J., Prentice-Hall. 322p.

Vukobratovic, M., Hristic, D. and Stokic, D. (1981) Dynamic Control of Industrial Manipulators. The Industrial Robot, June : 104-9.

Vukobratovic, M. and Stokic, D. (1982) A Procedure for the Interactive Dynamic Control Synthesis of Manipulators. IEEE Trans Syst. Man and Cybern., SMC-12(4) : 521-8.

Waltz, D.L. (1982) Artificial Intelligence. Sci American, October : 101-22.

Webb, J.C. (1980) Mechanism, Mentalism, and Metamathematics. An essay on Finitism. Dordrecht, D. Reidel. 277p.

Welford, A.T. (1980a) Reaction Times. editor. London, Academic Press. 418p.

Welford, A.T. (1980b) Choice Reaction Time : Basic Concepts. Chapter 3 in Welford (1980a). p.73-128.

Wexler, K.R. and Culicover, P.W. (1980) Formal Principles of Language Acquisition. Mass. MIT Press. 647p.

Whitney, D.E. and Junkel, E.F. (1982) Applying Stochastic Control Theory to Robot Sensing, Teaching, and Long Term Control. Proc. 12th Int. Symp on Industrial Robots; 6th Int. Conf. on Industrial Robot Technology. Paris. France. June. p.445-57.

Wiesendanger, M. (1978) Commentary in Roland. p.167.

Wilde, D.J. (1964) Optimum Seeking Methods. N.J., Prentice-Hall. 202p.

Winston, P.H. (1972) The MIT Robot. Chapter 24 in Machine Intelligence 7 edited by B.Meltzer and D.Michie. Edinburgh Univ. Press. p.431-63.

Winston, P.H. and Brown, R.H. (1979a) Manipulation and Productivity Technology. In Winston & Brown (1979b). p.211-8.

Winston, P.H. and Brown, R.H. (1979b) Artificial Intelligence: An MIT Perspective. Vol.2. editors. Mass., MIT Press. 486p.

Witten, I.H. (1976) Learning to Control Thesis: Ph.D.: Electrical Engineering Science. University of Essex.

Wu, C. and Paul, R.P. (1982) Resolved Motion Force Control of Robot Manipulator. IEEE Trans. Syst. Man and Cybern., SMC-12(3) : 266-75.

Young, J.F. (1973) Cybernetic Engineering. London, Butterworths. 153p.

Young, K.D. (1978) Controller Design for a Manipulator Using Theory of Variable Structure Systems. IEEE Trans. Syst. Man and Cybern., SMC-8(2) : 101-9.

Zecha, M., Fritzsch, K. and Schwarz, A. (1982) Second Generation Robots - Research and Some Applications in the GDR. Proc. 12th Int. Symp. on Industrial Robots; 6th Int. Conf. on Industrial Robot Technology. Paris. France. June. p.311-9.

GLOSSARY

ACS	(arm control system) The system in a robot's body to which commands are sent by the main controller of the robot. See the start of chapter II, Figure II-1, and Figure III-2.
action	A command sent to the robot's body control systems by the main robot controller. Also called a motor command. See the start of chapters II and III.
arm-robot	The simple real robot reported in chapter V. It comprises (a) a simple, single-degree-of-freedom arm, and (b) a simple MCLS. See the appendix of chapter V for a detailed description.
back-reflex	A variation of a reflex in which (a) the reflex-action is remembered as if it occurred before the reflex-trigger, and (b) the reflex-action is not performed, but only remembered. Back-reflex speech is sometimes called mimic-speech. See sections 3 and 1.3.1 of chapter VII and Figure VII-6.
C	Motor command, or action, stored during leading.
C'	Motor command, or action, to perform.
Cond	(see production). A set of stimuli fedback from the robot's body. See chapter III.
context	A set of recent stimuli and/or actions. A context is formed using a context template in a rule. A context is a subset of the set: [recent stimuli] u [recent actions], where u indicates set union. See section 2.1 of Chapter IV.
Cor	A correction action. A correction action for control system x in the robot's body is added to the action next performed by x, after the correction action is selected. See chapter III, particularly sections 1 and 2.
d	Discount factor. See section 3 and the start of chapter VI.
D(i,n+1)	The change in value of the context i between the nth and the (n+1)th leak-back step under decision $k'_b(i)$ selected at the nth step. See equation (8) in section 2.2 of chapter VI.

decision procedure — A decision procedure is an algorithm for selecting actions to perform from the combined predictions of the rules in an MCLS. See section 2.1 of Chapter IV.

discounting — The value of an expected future reward k steps away is reduced by d^k. d is the discount factor. Discounting may be thought of as representing either (a) the smaller real value of a reward unit obtained in the future, compared to one obtained now, or (b) the possibility that something will change the system being optimized. See chapter VI, especially section 3.

dual control — Robot actions are selected for dual, conflicting purposes: both (a) to "best" achieve a goal, given the known action effects; and (b) to find out more about action effects so that goals may be "better" achieved in future. "Best" may mean "gaining the greatest reward" or "in the fewest steps", etc. See section 4.2 of chapter III.

Goal — A code put into a production to indicate that it is a goal production. See section 2.1 of Chapter IV and section 4 of Chapter III.

grammar — A finite set of rules for specifying all the sentences in a language. See section 1 of chapter VIII.

GS system — (goal-seeking system) An ordered list of goals, plus a list of quintuples <Cond, C', <Next Cond, probability(Next Cond)>, <Goal>, Val> or for quintuples stored during leading <non-force Cond, <C, Next non-force Cond, <Goal>, Val>>. A GS system may also have in it a list of VC two-tuples. See Figures III-6 and III-10 and sections 1 and 4 in chapter III.

guiding method — The robot teaching method in which a keyboard or some other controlling device is used for teaching the robot. A large degree of real-time feedback is provided to the teacher, in contrast to the programming method. See section 3.2 of chapter II.

k_b — The set of $k_b(i)$. See section 2 of chapter VI.

$k_b(i)$ — The optimal action to perform in context i. See the start of chapter VI.

$k'_b(i)$ — The estimated optimal action to perform in context i. See the start of chapter VI.

k(i,x) — The xth action predicted by context i. See the start of chapter VI.

LTM	Long term memory. The network of productions stored in an MCLS's rules. See section 2.1 of chapter IV.
leading method	The teaching method whereby the teacher manually moves the robot through the movements required. Also called lead-through. See chapters III, IV and V, and section 3.1 of chapter II.
leak-back	The successive overrelaxation method of solving an MDP, GS system or rule, that is investigated in chapter VI. See section 2 of chapter VI.
MCLS	(multiple context learning system) One or more rules, a decision procedure plus a goal. See Figure IV-1 and section 2.1 of Chapter IV.
MDP	Markov decision process. An MDP has an internal representation of the same form as the internal representation in both a GS system and a rule. See Figures III-6 and VI-1 and the start of chapter VI.
motor command	A command sent to the robot's body control systems by the main robot controller. Also called an action. See the start of chapter II.
NHM, NHM' and NHM"	The negation handling MCLSs. See chapter IX.
negation problem	The problem of a robot doing something positive in the absence of a certain condition. See chapter IX.
PSC	(production system of corrections). An ordered list of actions, plus a list of three-tuples, <step, Cond, C'>. A PSC is added to a list of two-tuples for VC. See Figures III-5 and III-9, and sections 1 and 3 of chapter III.
PSC-M&A	An advanced PSC with a production system for movements and a production system for actions. See appendix to chapter III, section A.4.
prediction	See section 2.1 of Chapter IV. Of a production: an action, stimulus or action-stimulus that is the second member of a production. Of a rule: the prediction of the production whose context matches the current context.
priority	The weightiness of a rule's predictions in the decision procedure. See section 2.1 of chapter IV.

prob or p — Probability. For example, prob(Next Cond) is the probability that Next Cond will occur, and p(i,j,k(i,x)) is the probability that the next context after i will be j if action x is performed. See GS system and rule.

production — A three-tuple of a PSC, a quintuple of a GS system, or a quintuple, three-tuple, or two-tuple of a rule.

programming method — The robot teaching method in which a keyboard or some other device is used for teaching the robot. Little or no real-time feedback is provided to the teacher, in contrast to the guiding method. See section 3.2 of chapter II.

q(i,k(i,x)) — The reward obtained immediately if action k(i,x) is performed in context i. See the start of chapter VI.

reflex — The preprogrammed triggering of a particular action, a reflex-action, by a particular stimulus, a reflex-trigger stimulus or just reflex-trigger. See section 1 of chapter VII.

reward — A goal with a size.

rule — A network of productions having one of the five types (i) to (v) below, a template for forming the productions, and, if the productions are of type (iii) or (iv), an algorithm for calculating Val in each production. See section 2.1 of Chapter IV.

(i) <Context, action>
(ii) <Context, action, Goal>
(iii) <Context, action, Next Context, Goal, Val>
(iv) <Context, action, <Next context, prob(Next context)> Goal, Val>
(v) <Context, prediction, Goal>

Sp — Speech words

STM — Short term memory. The current contexts of a robot with an MCLS. See section 2.1 of chapter IV.

state — The state of an MDP is equivalent to both the Cond of a GS system, and the context of a rule. See the start of chapter VI.

step number — A sequence number associated with each motor command in the motor command sequence of a PSC.

stimulus	Preprocessed information from the robot's sensors. For example, the fedback arm angle, and the weight sensed on the wrist may both be stimuli.
TM	Turing Machine. A finite automaton connected to an unbounded tape memory. See section 2.2 of chapter VIII.
template	A prescription for forming the context - prediction pairs in productions. See section 2.1 of chapter IV.
transition	In a GS system production : prob(Next Cond), Next Cond. In a rule : prob(Next Context), Next Context, or just Next Context. See section 2.1 of chapter IV and section 4 of chapter III.
UTM	Universal TM. A simple TM that can simulate any other TM. See section 2.2 of chapter VIII.
v(i)	The optimal Val of context i. See the start of chapter VI.
v'(i)	The estimated optimal Val of context i. See the start of chapter VI.
v(i,k(i,x))	The Val of context i if action k(i,x) is performed. See the start of chapter VI.
v'(i,k(i,x))	The estimated Val of context i if action k(i,x) is performed. See the start of chapter VI.
v'(i,n)	The estimated v(i) on step n of leak-back. See section 2 of chapter VI.
VC	(verbal correcting) A method by which short verbal phrases invoke motor command corrections to the motor commands a robot is performing. The internal representation is a list of two-tuples, <Sp, Cor>. See Figure III-4 and sections 1 and 2 of chapter III.
Val	A measure of how worthwhile it is for the robot to reach that production, in seeking the currently active goal through the network of productions. The greater the Val is the closer the production, with that Val, is to the active goal. See section 2.1 of chapter IV and section 4 of chapter III.

APPENDIX A

HUMANS AND ROBOT DESIGN

Some comments on human acquisition and performance of motor tasks are given in section 1. Some comments on human muscles and eye movements are given in section 2. These comments are kept separate from the rest of the thesis in order to emphasize that my aim is to enhance the design of teachable adaptable robots, not to design a robot to mimic humans. The little we know about human acquisition and performance of tasks is a source of ideas for robot design.

A.1 HUMAN ACQUISITION AND PERFORMANCE OF MOTOR TASKS

1. Muscles behave like springs at low disturbing frequencies. The spring coefficient is approximately a linear function of the level of tonic (static) contraction (Benati et al, 1980). At "presumed physiological muscle lengths" an active muscle fibre will increase its contractile tension when stretched and an inactive fibre will produce negligible tension, so the greater the number of active muscle fibres, the greater the muscle stiffness. Muscle itself acts like a spring, even without any spindle feedback (Sakitt, 1980).

Neuromuscular spindle fibres provide feedback of muscle length and rate of change of muscle length (Gowitzke and Milner, 1980). Thus velocity and position feedback are provided for muscle control. The gains of both these feedback loops can be altered. By controlling the bias or rest length of position feedback muscle spindles the muscle can be servoed to a particular muscle length (Gowitzke and Milner, 1980; Stark, 1968; van Dijk, 1978a). Note that a muscle spindle gives feedback signals only when it is stretched past its rest length.

Spindles do not respond at all when they are shorter than their rest length. So a muscle can be servoed only in one direction by its own spindles. Opposing muscles and spindles are used for stretching a muscle. A joint can be servoed in both directions by controlling the biases of opposing muscles' spindles. Gamma motoneurons control the rest lengths of muscle spindles.

It is not clear how servo control of spindle rest lengths and direct control of the muscle fibres are combined for performing movements (van Dijk, 1978a; 1978b; Stark, 1968; Gowitzke and Milner, 1980; Stein and Oguztoreli, 1981).

2. The arm-robot MCLS of chapter V does not compensate for disturbances during the course of an action. It does not change an action part way through. However the arm-robot can compensate for disturbances by doing different sequences of actions. These actions must have been acquired beforehand. For example, the arm-robot is able to cope with the sagging of the weight, as discussed in chapter V. The arm-robot did a sequence of actions that compensated for the sagging caused by the weight.

In humans the latency of sensory feedback about fast movements may be too great for any motor "reprogramming" to occur at high levels during the movements (Roland, 1978; Pribram, 1978; Stein, 1978; Iannone, 1978). Roland suggests that perhaps the role of peripheral information is to update the cerebral cortex with data about the consequences of voluntary contractions. Sensory feedback of tension and voluntary muscular movement probably affects only slow movements. Experiments indicate that the major function of sensory feedback is to update and adjust learned motor programs rather than assist motion execution (Kelso, 1978). "Recently the importance of proprioceptive signals in the adaptive modification of control programs through a process of gain changes as well as changes in the coupling among active

muscles has been stressed" (p.554 Bizzi et al, 1978). Proprioceptive signals are movement feedback signals, for example the spindle feedback signals (Gowitzke and Milner, 1980). Sensory feedback has two roles (Roland, 1978). It provides feedback to subcortical parts. It updates cerebral cortical information concerning the consequences of voluntary contraction in the periphery. This updating is for the programming of further voluntary actions. In learning to use a control stick to reproduce a curve on an oscilloscope, "Subjects need information which they can use to improve their performance on subsequent trials, not information that contributes to better performance on the trial in progress" (p.398. Pew, 1974).

3. Monkeys are totally impervious to peripheral information when advance information regarding movement is available. The evidence for humans is the same (Kelso, 1978). The cerebellum can coordinate movement even in the absence of all information from the periphery of the body (Llinas, 1975). Almost all central motor program generators can operate without the need for peripheral sensory input (Selverston, 1980). Cutaneous afferent information is important in motor control (Duysens and Loeb, 1978). Just because humans can do motor tasks without proprioceptive feedback doesn't mean that they don't normally use it (Wiesendanger, 1978).

The arm-robot relies on predictions from ANGLE productions and ACTION productions for doing actions. A prediction from an ACTION production is not enough. However it is possible to have an MCLS that can perform actions on the basis of productions that do not include peripheral, or sensory information about the results of those actions. Some other types of production could be added to the arm-robot MCLS, for example productions which predicted arm actions from contexts with speech events in them [see Andreae (1980a; 1980b) where hand moves are predicted by contexts that don't contain information about the position

of the hand.].

4. I have called the arm of the arm-robot "relaxed" when in its training mode with the position error signal removed.

Gowitzke and Milner (1980) report these facts. If a muscle is "resting" and is passively moved then there is a short period of resistance followed by a readjustment of gamma motoneurons to the new position. When a muscle is at rest, gamma bias is held just below firing level. If a muscle is shortened while at rest then there is no response from its spindles. But soon the original rest firing rate resumes. The slack is always taken up.

I mentioned in 1. that a muscle spindle gives a feedback signal only when stretched past its rest length. So the arm may be relaxed by setting the gamma biases of opposing muscles so that no spindles are responding. That is, none of the spindles are stretched.

5. I suggested in section A.4.2 (a) of chapter V that the gain of the arm control system be adjusted in proportion to estimates of the moment of inertia of the arm and load. Acceleration measurements and torque signals would be used for estimating the moment of inertia, according to equation (11) in chapter V.

Man can adjust his commands to motoneurons in proportion to the "requirements of expected performance" (Roland, 1978). Joint receptors provide position, velocity and acceleration information (Gowitzke and Milner, 1980). Roland (1978) presents evidence for there being both a "sense of effort" and a "sense of force". That is, information about the actual force exerted is available and so is information about the force requested by higher levels of the brain. It has been suggested that the stretch reflex---the position error feedback provided by spindles---is a test signal and that the central nervous system uses the position error information to control force (Arbib, 1979).

Benati et al (1980) note that the muscular output force and muscular impedance can be programmed independently: the output force by the imbalance between opposing muscle excitations, and the impedance, or compliance, by the level of coactivation of opposing muscles. As noted in 1., the gains of position and velocity feedback loops can be controlled. Dyhre-Poulsen (1978) notes that one can change a subject's response to unexpected disturbances during movements by modifying the subject's instructions.

The firing rate of joint position receptors will change if the joint angle changes by more than two degrees (Gowitzke and Milner, 1980). Sakitt (1980) suggests that there are only twenty independent positions the forearm can be put in.

6. I mentioned in section A.4.3 of chapter II that integral error control could be added to a second order arm. This would eliminate static errors in position, such as those caused by gravity.

Humans do not seem to have such compensation in their muscle control systems. Bizzi et al (1978) have attempted to answer the question "How effectively do primates compensate when their movements are met by unexpected load disturbances?". They discovered that "reflexes" provide only 10% to 30% of perfect compensation. By "reflexes" they mean all automatic compensation systems not including the compensation provided by the inherent elasticity of the actual muscles.

A.2 HUMAN MUSCLES AND EYE MOVEMENTS

1. It is reported (Stark, 1968) that the human eyeballs' dynamics are those of a second-order system with a damping factor of 0.7 and a natural frequency of 240 radians per second. A second order system is shown in note 7 of chapter VII.

However, it is not certain that the eye-muscle system exhibits the "stretch-reflex" that is characteristic of a position feedback system and is exhibited by other muscle systems in humans (Lippold, 1973). The stretch-reflex is the tendency for a position feedback systems's output to be independent of the external load applied to it (Lippold, 1973). It is the direct result of negative feedback, an error signal between output and input driving the system in opposition to external forces.

Since the eye does not seem to exhibit the stretch reflex when an external load is applied to it, Lippold has suggested that the position feedback loop is not present in humans, except for a rather crude feedback path which keeps the eyeball in its "bony orbit".

On the other hand, it seems that one can perceive the position of one's eye, while in a darkened room, if controlled movements are introduced in a sensible manner (Granit, 1978; Shebilske, 1978).

For a robot eye control system the following points are important:-

(i) Negative feedback of position information has clear advantages in terms of the mechanical design of the eye movement system (Di Stefano et al, 1967).

(ii) Position feedback from the eye-muscle system might be used when the eye is moved (Lippold, 1973).

Note that once a visual system for a robot has been developed, it may become obvious that the eye-muscle system should or should not have position feedback.

2. There is evidence (Fender, 1964a; 1964b; Lippold, 1973) that displacement, and perhaps velocity information is fed back from the retina to the eye control system.

3. There are four types of eye movement:-

(i) Saccadic: a rapid shift of the angle of "gaze" (Bahill & Stark, 1979; Arbib, 1972).

(ii) Smooth pursuit or tracking: the eye follows a slowly moving target (Bahill & Stark, 1979; Arbib, 1972).

(iii) Vergence: slow smooth movement of the eye between targets (Bahill & Stark, 1979).

(iv) Vestibulo-ocular: eye movements to maintain visual stability during head movements (Bahill & Stark, 1979).

4. Peripheral visual information is used to direct saccades (Bahill & Stark, 1979).

5. A different part of the brain is involved in the production of saccades to that which is involved in producing smooth tracking movements. This suggests that two separate systems generate these two types of eye movement (Arbib, 1972; Stark, 1968; Fender, 1964b).

6. There is evidence of visual information being used at a low level in the hierarchy of the control of eye movements (Fender, 1964a). This hierarchy extends from the cortex down to the eye muscles. In particular (Fender, 1964a),

(i) velocity signals are available at a retinal level for "targets" in the periphery, but not for those in the very centre of the visual field,

(ii) "image-displacement", that is position, information about the central, or foveal, visual field seems to be retinal in origin.

Fender (1964a) also points out that the retina is part of the brain that became detached in the course of evolution.

7. There is evidence (Pribram, 1969) in favour of a servomechanism type of reflex organization rather than a "reflex arc" type of organization. A reflex arc type of organization would have reflex

movements initiated by stimuli, rather than the stimuli affecting control signals from higher up in the brain. The reflexes discussed in chapter VII initiate actions that are learned by the MCLS. This form of reflex should not be confused with the effects that servomechanism effectors have. For example, the "stretch reflex", which I mentioned in 1. is not the sort of reflex I discuss in chapter VII.

8. It has been suggested (Stark, 1968; Crossman & Goodeve, 1963) that eye control in humans is discrete.

9. There is some evidence (Noton & Stark, 1971) that the human visual processing system processes information serially by scanning the fovea over the visual field. The fovea is the central part of the retina which provides visual detail.

10. Note that a linear reciprocal muscle pair is the same as an ideal, second-order servomotor or a single linear muscle that can contract and expand. A reciprocal muscle pair is two opposing muscles that can contract but not actively expand, as shown in Figure VII-20 (b) of chapter VII. The equivalent control signal for the single component system is the difference between the two control signals of the muscle pair. Of course the single component system must have the same damping factor and natural frequency as the muscle pair, for this equivalence to hold.

11. Retinal cells are much more closely packed in the fovea than in the periphery (Tolkmitt, 1976), suggesting that much more detailed information is sensed there.

12. Although the brain processes events serially, it has many parallel-acting neural circuits (Arbib, 1972). The control systems in this thesis are for a robot and so they have been developed with serial digital computation in mind.

APPENDIX B

FUTURE RESEARCH

B.1 SUGGESTED POST GRADUATE PROJECT FOR THE ELECTRICAL AND ELECTRONIC ENGINEERING DEPARTMENT, UNIVERSITY OF CANTERBURY

Arm project: Build a robot that can be taught to do tasks with an arm; for example, (a) the task of moving some simple blocks around so that a particular configuration of blocks is achieved, (b) the task of moving a number of objects from a machine to a conveyor belt, avoiding various obstacles, and (c) the task of stacking blocks on a moving conveyor belt or on a rotating turntable.

The Arm project could involve:

1. Make a controller that accounts for the dynamics of the arm.

2. Make a system to alter the feedback gains of the arm controller so that the arm remains stable when there are loads in its hand.

3. Make a system to account for the dynamic effects of loads.

4. Make the arm relaxable and implement the leading method on the arm, so that the arm can be taught movements by being led.

5. Make a sensor for the gripper. This sensor might be used to trigger a grasp reflex.

6. Make a vision system that gives the robot the information it needs for coordinating its hand movements with the objects in the world. For example, to enable it to avoid obstacles.

7. Make a proximity sensor for the arm that tells the learning system when any part of the arm is close to an object.

8. Connect up speech to the arm MCLS so that a teacher can (a) verbally correct the robot's movements, (b) verbally reward the robot, and (c) do experiments with language.

9. Make an MCLS for the arm and controller, vision, touch, speech, etc, which enables the tasks, (a), (b) and (c) above to be learned.

10. Make a learning system comprising a production system of corrections (PSC).

11. Make a learning system comprising a goal-seeking (GS) system.

B.2 SUGGESTED LONG TERM RESEARCH AND DEVELOPMENT PROGRAM.

1. Make a robot arm that is leadable, light enough to be easily led, and powerful enough to move as rapidly as human arms can. For an electric arm to be leadable, the gears and motor must be ones that move when you move the arm. Non-reversible gears, for example worm drives, cannot be used.

2. Make a limited vocabulary, fast speech recognition system.

3. Use the speech system with the arm to investigate the verbal correcting (VC) scheme. In particular the 'naturalness', speed and convergence of VC should be investigated.

4. Add a production system of corrections (PSC), and investigate in particular its 'naturalness', and its ability to provide the conditional branching required in both industrial and other "sequence of movements" tasks.

5. Extend the PSC to a PSC-Movements and Actions (PSC-M&A).

6. Add a goal-seeking (GS) system to the robot and VC scheme. Investigate the robot's ability to learn goal-like tasks.

7. Design an MCLS that is suitable for the robot arm. It must (a) enable tasks learnable by the systems above to be taught, and (b) enable some additional tasks to be taught. The teaching must be 'natural'.

At steps 3 through 7 a robot system should be completed, and used for the tasks it is most suited to.