1

MicroD

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R S C 1970; c C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c C-30.

Some Learning Applications To Robotic Motion Problems

by

María Nicté Andrade Abularach,
Bachelor of Systems Engineering

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Computer Science—

School of Computer Science
Carleton University
Ottawa, Ontario

26th November, 1987

© Copyright
1987, María Nicté Andrade Abularach

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-40620-8

The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

**SOME LEARNING APPLICATIONS TO ROBOTIC
MOTION PROBLEMS**

submitted by María Nicté Andrade Abularach,
Bachelor of Systems Engineering
in partial fulfillment of the requirements for
the degree of Master of Computer Science

Thesis Supervisor

Chairman, School of Computer Science

Carleton University

Date

Abstract

A basic problem in Robotics is that of planning motions to solve some previously specified task, and then controlling the robot by specifying the primitive commands necessary to execute those motions. The paths followed by the robot are called trajectories, and usually consist of a sequence of desired positions, velocities and accelerations of the manipulator's end effector.

The problem of trajectory planning is well solved for the noise-free environment. In this thesis we study various methods of solving motion problems for manipulators in noisy workspaces. Apart from reviewing previous methods, a new algorithm is presented using stochastic automata, which are used to determine the trajectory of the robot. The implementation of some previous methods and experimental results of the new algorithm are given.

We also study the motion planning for an autonomous robot in an unknown terrain. This problem, which involves the model acquisition and learning stages, has also been investigated.

Acknowledgements

I am grateful towards God and my family, for all the moral support required in order to complete the Master program.

I wish to thank my supervisor Professor B. John Oommen, who has devoted a lot of time in discussions and verification of this thesis.

I take this opportunity to thank my husband, Roland, for his assistance and advice throughout my studies, and for all of those who have helped in the proof-reading, especially Nicole and Peter.

Finally, I would like to thank Carleton University and the Fondation Yvon Boulanger for the financial support throughout the entire program.

Table of Contents

Acceptance Sheet.....	ii
Abstract	iii
Acknowledgements	iv
Table of Contents.....	v
List of Tables	viii
List of Figures	ix
1. Introduction	1
1.1 Thesis Objective	2
1.1.1 Problem Statement for Manipulator Arms	3
1.1.2 Problem Statement for Autonomous Mobile Robots	3
1.2 Thesis Outline	3
2. General Descriptions and Definitions	6
2.1 Robot Anatomy	6
2.2 Motion	7
2.2.1 Motion Types	9
2.3 Control Systems	10
2.3.1 Basic Control Systems	11
2.4 Compliance	13
2.5 Kinematics	13
2.5.1 Direct Kinematics	14
2.5.2 Reverse Kinematics	16
2.6 Dynamics	19
2.7 Sensors	20
2.8 Precision of Movement	20
2.8.1 Position Errors	22
2.9 Conclusion	23
3. Previous Methods	24
3.1 Brief History on Trajectory Planning	24
3.2 On-line Cubic Splines for Joint Trajectory Interpolation	27
3.2.1 Experimental Results	32
3.3 Brief History of Control Algorithms	36
3.4 Miller's Algorithm	39
3.4.1 The CMAC Module	40
3.4.2 CMAC-Based Control	41

3.4.3 Static Positioning Experiment	42
3.4.4 The Conveyor Tracking Experiment	43
3.5 Betterment Process	44
3.6 Conclusion	48
4. Methods Based on Learning Automata	49
4.1 Learning Automata Models	49
4.1.1 Random Environments and Learning Objective	49
4.1.2 Learning Criteria	51
4.1.3 Classification of Learning Automata	51
4.2 Fixed Structure Stochastic Automata	52
4.2.1 Krinsky Automaton	53
4.3 Variable Structure Stochastic Automata(VSSA)	55
4.3.1 Linear Reward-Penalty Scheme	57
4.3.2 Linear Reward-Inaction Scheme	58
4.4 Discretized Learning Automaton	59
4.4.1 The Two-Action DLPP Automata	60
4.4.2 Multi-Action DLPP Automaton	69
4.4.2.1 The Three-Action DLPP Automata	74
4.4.3 Discretized Linear Reward-Inaction Automata	76
4.4.3.1 Markovian Representation of the DLPI Automaton..	78
4.5 Manipulator Control Using Automata	79
4.6 Experimental Results	82
4.7 Conclusions	106
5. Autonomous Mobile Robots	108
5.1 Chapter Statement	108
5.2 Literature Review and Previous Solutions	108
5.3 Algorithm due to Oliya and Ozguner	111
5.3.1 Region Classification	111
5.3.2 Path Planning	112
5.3.3 Path Smoothing	113
5.4 Mitchell's Algorithm	115
5.4.1 Implementation	117
5.5 Robot Navigation Algorithm Using Learned Spatial Graph.....	122
5.5.1 Local Optimization of Object Avoidance	122
5.5.2 Terrain Model	125
5.5.3 Path Planning and Learning	126
5.6 Conclusion	130
6. Path Planning with Learning	131

6.1	Notations and Definitions	131
6.2	Local Navigation and Learning	132
6.3	The Backtracking Facility	136
6.4	Global Navigation	137
6.5	Implementation and Results	138
6.6	Conclusions	149
7.	Conclusions	150
7.1	Summary of the Results	150
7.2	Future Research	152
	References	154

List of Tables

Table	Description	Page
4.1	The Transition Matrix of the Three-Action DLPP Automata for $N=1, \dots$	76
4.2	Results obtained for the Horn's 2-DOF Robot, in which each automaton had three actions and all the automata received a single global response.....	84
4.3	Results obtained for the Horn's 2-DOF Robot, in which each automaton had three actions and all the automata received the responses as given in the column headings.....	84
4.4	Results obtained for the Horn's 2-DOF Robot, in which the DLPP automata had two actions and the automata received the responses as given in the column headings.....	85
4.5	Results obtained for the Horn's 3-DOF Robot, in which the DLPP automata have Two and Three-Action as given in the column headings	87
5.1	Typical output of the classification obtained using the Laser Rangefinder	112
6.1	Expected number of vertices and edges learned using random goal points for the obstacle terrain shown in Figure 6.1a.....	148

List of Figures

Figure	Description	Page
2.1	Basic Control System	12
2.2	Direct or Forward Kinematics of Horn's 2-DOF Robot.....	15
2.3	Inverse or Reverse Kinematics of Horn's 2-DOF Robot	17
2.4	Forward Kinematics of Horn's 3-DOF Robot	19
2.5	Repeatability and Accuracy	22
3.1	Sample Points on a Joint Trajectory	28
3.2.a	Expected Trajectory of Horn's Robot using the Cubic Spline Interpolation technique described in Section 3.2. The details of the initial and final configurations are given in Section 3.2.1.....	34
3.2.b	Expected change in distance for the scheme described in Section 3.2 from the initial configuration to the final mean configuration The set-up is described in Figure 3.2a.....	35
3.3	C ¹ - Type Betterment Process	46
4.1	The Automaton-Environment Interaction	50
4.2	Z _{2N,2} : The Two-Action 2N-State Krinsky Automaton	54
4.3	Transition map of the Two-Action DL _{RP} Automaton	61
4.4	The Equilateral Triangle Representation of the Three-Action DL _{RP} Automata for N=1	75
4.5	The Two-Action DL _{RI} Automaton	77
4.6	Manipulator Motion Planning for Horn's 2-DOF Robot	81
4.7a	Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Two-Action Krinsky automata with $RN = 8$ and with a single response controlling both the automata	89

4.7b	Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.7a.....	90
4.8a	Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Two-Action LP_I automata with $a = 9$ and with a single response controlling both the automata	91
4.8b	Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.8a.....	92
4.9a	Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Two-Action LP_P automata with $a = 0.9$, $b = 0.5$ and with a single response controlling both the automata	93
4.9b	Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.9a.....	94
4.10a	Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Two-Action DL_{PI} automata with $NR = 12$ and with a single response controlling both the automata	95
4.10b	Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.10a....	96
4.11a	Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Two-Action DL_{PP} automata with $NR = 6$ and with a single response controlling both the automata.	97
4.11b	Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.11a....	98
4.12	Expected number of iterations for the robots to enter the three standard deviation disk of the final point for cases when the automata use a single response.....	99
4.13a	Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Two-Action DL_{PP} automata with $NR = 9$ and with a single response controlling both the automata.....	100

4.13b	Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.13a...	101
4.14	Expected number of iterations for the robots to enter the three standard deviation disk of the final point for cases when the automata use a single response.....	102
4.15a	Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Three-Action DLPP automata with NR=9 and with separate response for each automata, where these responses are obtained using a first order approximation of inverse kinematics...	103
4.15b	Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.15a..	104
4.16a	Expected Path (given as a sequence of points) of the Three-link Horn's Robot with links of length unity. The control is achieved using the Three-Action DLPP automata with NR = 9 and with a single response controlling both the automata.....	105
4.16b	Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.16a...	106
5.1	An implementation of Mitchell's [62] algorithm. The robot is represented by the larger circle, the goal is represented by the smaller circle.....	119
5.2	Path taken by the robot from start to a subgoal position for the set-up described in Figure 5.1.....	120
5.3	Path taken by the robot from start to the goal position for the set-up described in Figure 5.1.....	121
5.4	First Case for Local Optimization of Iyengar's <i>et. al</i> Algorithm.....	123
5.5	Second Case for Local Optimization of Iyengar's <i>et. al</i> Algorithm. *	124
6.1a	The obstacle terrain for the experiments. The robot is represented by the larger circle, the goal position is represented by the smaller circle	139
6.1b	First Path taken by the robot in the set-up of Figure 6.1a.....	140

6.1c	The LVG after executing the Navigate-Local and the motion described in Figure 6.1b.....	141
6.2a	The robot is represented by the larger circle, and the second the goal position is represented by the smaller circle	142
6.2b	Second Path taken by the robot in the set-up of Figure 6.2a.....	143
6.2c	The LVG after executing the Navigate-Local and the motion described in Figure 6.2b.....	144
6.3a	The robot is represented by the larger circle, and the third the goal position is represented by the smaller circle	145
6.3b	Third Path taken by the robot in the set-up of Figure 6.3a.....	146
6.3c	The LVG after executing the Navigate-Local and the motion described in Figure 6.3b.....	147
6.4	Graphical plot of the Table 6.1.....	149

Chapter 1

Introduction

The integration of robots into the production picture is one of the most promising ways of using technology to increase output per man-hour and thereby to improve productivity. For many industrial operations, robots may be the cheapest, if not the only way to comply with safety regulations. Because the industrial robot is immune to many workplace hazards, it can present a cost-effective method of complying with regulations. For example, by using robots in welding processes, manufacturers can eliminate the need for expensive external gear and the ventilation equipments necessary to protect welders from the hazards of grinding, hammering, and other welding processes. Savings in safety equipment is not the only contribution that robots can make. Rising wage rates definitely justify the capital investments necessary to obtain the robots.

Since automated production systems of the future will rely heavily on robots, design engineers will spend a significant amount of time searching for the most appropriate robot for a given work environment or task. Robotic research takes many directions and is prompted by a wide variety of goals. These goals include both the development of mobile autonomous robots capable of understanding and responding to new environments and also the development of sophisticated flexible, reprogrammable, computer controlled machine tools. This variety of goals and intended applications creates a corresponding variety of motion related problem areas. For example, a mobile robot cannot reasonably be expected to have an accurate detailed model of its environment. On the other hand, a robot designed to be used on an automated assembly line may have very detailed information from a CAD/CAM database

about the objects it manipulates, and the motion constraints are thus correspondingly different from the former case.

Some of the areas in which researchers are studying problems in robotics include the design of robotic languages, trajectory planning, motion planning in the presence of obstacles, and strategies for fine motion control. Motion planning, i.e., the problem of computing a path for a robot, is recognized as a fundamental problem in robotics. A typical formulation of this problem is as follows: given a robot and a description of its cluttered environment, how can we plan a collision free path of the robot between two specified positions. It is not surprising that the motion planning problem has been addressed in several disciplines and is also called the findpath problem, the obstacle-avoidance problem, the collision-avoidance problem or the piano movers' problem. An important source of inspiration for research on algorithms for motion planning has come from the need to solve path finding problems for mobile robots.

There are two archetypes of robots currently in practice: one is a multi-jointed robot manipulator arm fixed to its base, and the other is an autonomous mobile robot. Although the engineering issues for the arm and the mobile automaton are different, their motion planning problems are generally studied with a common underlying mathematical framework.

1.1 Thesis Objective

The objective of the thesis is to study various methods of solving motion problems for manipulators in noisy workspaces as well as the motion planning problem for an autonomous robot operating in an unknown terrain. Apart from reviewing previous methods, a new algorithm is presented using stochastic automata. These learning automata are used to determine the trajectory of the robot in Cartesian space. We also study motion planning for

an autonomous robot in an unknown terrain. This problem involves activities such as model acquisition and learning by sensing.

1.1.1 Problem Statement for Manipulator Arms

We consider the problem of a robot manipulator operating in a noisy workspace. The robot is assigned the task of moving from P_i to P_f , where P_i and P_f are the initial and final positions and orientations of the robot. Since P_i is its initial position, this position can be known fairly accurately. However, since P_f is usually obtained as a result of a sensing operation, possibly vision sensing, we assume that P_f is noisy. We propose a solution to achieve the motion which involves learning automata. The strategy we propose does not involve the computation of any inverse kinematics. An automaton is positioned at each joint of the robot, and by processing repeated noisy observations of P_f the automata operates in parallel to control the motion of the manipulator.

1.1.2 Problem Statement for Autonomous Mobile Robots

The second major problem we shall study involves that of navigating an autonomous mobile robot through an unexplored terrain of obstacles. The process of navigation constitutes a learning phase in which the robot learns the information about the obstacle terrain, and the path planning phase in which the path of the robot is actually computed.

1.2 Thesis Outline

In Chapter Two we outline some of the basic concepts and definitions that will be used throughout the thesis. These concepts include terms such as robot anatomy, motion, control systems, compliance, kinematics, direct kinematics and reverse kinematics. Basic terminology involving dynamics,

sensors, and errors will also be introduced. An example of a simple robot, namely Horn's robot operating in the plane will be presented to highlight some of these concepts.

In a general robot task, the end effector is required to move from one point to another in a manner specified by the operator. During these moves the robot must avoid collisions with obstacles, and the end effector may also be required to trace a specified path. The robot path specification is frequently done by requiring that the end effector pass through a number of intermediate points between the initial and final points. There are two methods of moving a manipulator between the two points: joint interpolated motion and Cartesian motion. The variations of these schemes is extensive. For example, either of these methods may be executed in a continuous path mode, in which a constant velocity is maintained along the trajectory. In such a case, the constant velocity causes problems in the specified motions because of the necessity of an instantaneous acceleration at each point. An alternative strategy in Cartesian motion without continuous path is that of requiring that the robot positions and velocities are specified at many intermediate points. This method is suitable for many applications but may be detrimental in some others such as spray painting and close-tolerance machining. A trajectory generation scheme based on spline functions does not have some of the problems of the above continuous path motions. In this thesis we study the problems of planning trajectories using various polynomials and of controlling the path using some closed loop control systems. The results obtained in this context are found in Chapter Three.

In Chapter Four we give a brief overview of the area of learning automata. In this chapter, the concepts of learning automata, Fixed Structure Stochastic Automata, Variable Stochastic Automata and Discrete Learning Automata are reviewed. We present the solution to the problem stated in

Section 1.1.1 by using Learning Automata. Experimental results are presented, indicating that the DLGP scheme is possibly the best automata for this problem.

An autonomous mobile robot may be characterized as a machine capable of motion planning execution and learning. Although there have been numerous efforts to design automated mobile robots, the work done with respect to robot navigation in an unexplored terrain is still in its infancy. In Chapter Five of this thesis we study and review the proposed algorithms of two groups of researchers.

In Chapter Six we present a path planning algorithm for the process of navigation in a completely unexplored obstacle terrain. This involves both learning the information about the obstacle terrain and path planning. The navigation process consists of a number of traversals; each traversal is from an arbitrary source point to an arbitrary destination point. At any stage of learning the partially learnt terrain model is represented as a Learned Visibility Graph, and it is updated after each traversal. This algorithm, which was primarily presented in a theoretical form, was also simulated. The results of the simulation are presented in Chapter Six.

Chapter Seven concludes the thesis with some overall remarks and suggests various areas for further research.

Chapter 2

General Descriptions and Definitions

In this section we shall describe some of the basic concepts and definitions that will be used throughout the thesis. These include the anatomy of a robot, and terms such as control systems, compliance, kinematics, direct kinematics and reverse kinematics. We will also discuss some basic terminology involving dynamics, sensors, and errors. An example of a simple robot, namely Horn's robot operating in the plane, will be presented to highlight some of these concepts. A number of references are available in this regard. In particular we refer the reader to the book by Brady, *et. al* [18], Groover, *et. al* [35], Paul [79], and Shahinpoor [84].

2.1 Robot Anatomy

Robot anatomy is concerned with the physical construction of the body, arm and wrist of a machine. Most robots used in plants today are mounted on a base which is fastened to the floor. The body is attached to the base and the arm assembly is attached to the body. At the end of the arm is the wrist. The wrist consists of a number of components that allow it to be oriented in a variety of positions. Relative movements between the various components of the body, arm, and wrist are provided by a series of joints. These joint movements usually involve either rotating or sliding motions, which will be described later. The body, arm, and wrist assembly is sometimes called the **manipulator**.

Attached to the robot's wrist is a hand. The technical name for the hand is the **end effector**. The end effector is not considered as part of the robot's anatomy. The arm and body joints of the manipulator are used to position and orient the end effector.

Industrial robots are available in a wide variety of sizes, shapes, and

physical configurations. The majority of today's commercially available robots possess one of five basic configurations:

A **Spherical** (or Polar) Configuration Robot uses a telescoping arm that can be raised or lowered about a horizontal pivot. The pivot is mounted on a rotating base. These robots perform two rotary motions and a single sliding (telescopic) motion.

A **Cylindrical** Configuration Robot uses a vertical column and a slide that can be moved up or down along the column. These robots can normally achieve both linear orthogonal motions and a rotary motion.

A **Cartesian** (or Rectangular Coordinate) Configuration Robot uses perpendicular slides to achieve motion in the x, y, and z axes. Movements normally consist of three linear orthogonal displacements along joint axes. The best resolution and accuracy within the robot's workspace are achieved with this configuration.

Revolute (or Articulated Jointed-Arm) Configuration Robots are humanoid type robots with redundant degrees of freedom. They consist of straight components connected by rotary joints.

The **Snake-like** (or Tensor-Arm) Robots consist of arms that can take any shape in three dimensional space. They consist of many elements that can move with respect to their adjacent elements.

2.2 Motion

Industrial robots are designed to perform productive work. The work is accomplished by enabling the robot to move its body, arm, and wrist through a series of motions and positions. Robot movement can be divided into two general categories: arm and body motions, and wrist motions. The individual joint motions associated with these two categories are sometimes referred to by

the term "Degrees of Freedom" (DOF) and a typical industrial robot is equipped with 4 to 6 degrees of freedom.

The robot motion is accomplished by means of powered joints. Three joints are normally associated with the action of the arm and body, and two or three joints are generally used to actuate the wrist. Rigid members, referred to as links, connect the various manipulators joints. In any link-joint-link chain, we shall call the link that is closest to the base in the chain the input link. The output link is the one that moves with respect to the input link.

The joints used in the design of industrial robots typically involve a relative motion of the adjoining links that is either linear or rotational. Linear joints involve a sliding or translational motion of the connecting links. This motion can be achieved in a number of ways by means of, for example, a piston, a telescoping mechanism, or relative motion along a linear track or rail. Our concern in this thesis is not with the mechanical details of the joint, but rather with the relative motion of the adjacent links. The term "prismatic joint" is sometimes used in the literature instead of the term "linear joint".

There are at least three types of rotating joints that can be distinguished in robot manipulators. In the first type, the axis of rotation is perpendicular to the axes of the two connecting links. The second type of rotating joint involves a twisting motion between the input and output links. The axis of rotation of the twisting joint is parallel to the axes of both links. The third type of rotating joint is a revolving joint in which the input link is parallel to the axis of rotation and the output link is perpendicular to the axis of rotation. In essence, the output link revolves about the input link as if it were in orbit.

The arm and body joints are designed to enable the robot to move an end effector to a desired position within the limits of the robot size and joint movements. For robots of polar, cylindrical, or jointed-arm configurations, the three degrees of freedom associated with the arm and body motions are:

- (i) Vertical traverse: This is the capability to move the wrist up or down to provide the desired vertical altitude.
- (ii) Radial traverse: This involves the extension or retraction of the arm from the vertical center of the robot.
- (iii) Rotational traverse: This is the rotation of the arm about the vertical axis.

Wrist movement is designed to enable the robot to orient the end effector properly with respect to the task to be performed. To solve the orientation problem, the wrist is normally provided with up to three degrees of freedom. The following is a typical configuration:

- (i) Wrist roll: Also called wrist swivel, this involves rotation of the wrist mechanism about the arm axis.
- (ii) Wrist pitch: Given that the wrist roll is in its center position, the pitch would involve up or down rotation of the wrist. Wrist pitch is sometimes called wrist bend.
- (iii) Wrist yaw : Given that the wrist swivel is in the center position of its range, wrist yaw is defined as right or left rotation of the wrist.

The reason for specifying that the wrist roll position be centered in the definitions of pitch and yaw is that rotation of the wrist about the arm axis will alter the orientation of the pitch and yaw movements.

2.2.1 Motion Types

There are three common types of motion that a robot manipulator can make in traveling from one point to another. These are slew motion, joint-interpolated motion, and straight line motion.

Slew motions represent the simplest type of motion. As the robot is commanded to travel from point A to point B, each axis of the manipulator

travels as quickly as possible from an initial position to the required final position. All axes begin moving at the same time, but each axis comes to a stop in an elapsed time that is related to the distance moved and the maximum allowable speed.

Joint-interpolated motion requires the robot controller to calculate the amount of time it will take each joint to reach a destination at the commanded speed. It then selects the maximum time among these, and uses this value as the time for all the axes. This means that a separate velocity is calculated for each axis, and that the joints start and stop their motions simultaneously.

Straight line interpolation motion requires the end effector of the manipulator to travel along a straight path defined in Cartesian coordinates. This is the most demanding type of motion for the controller to execute, except for a Cartesian coordinate robot. For manipulators with rotational joints, most straight line motions are unnatural and the controller must compute the sequence of incremental joint rotations required for the end of arm to move in a linear fashion. A generalization of this is the case when the manipulator has to go through a sequence of specified intermediate points, called VIA points.

2.3 Control Systems

The robot's capacity to move body, arm, and wrist is provided by the drive system used to power the robot. The drive system determines the speed of the arm movements, the strength of the robot, and its dynamic performance. In order to operate, a robot must have the means of controlling the drive system for properly regulated motion. Commercially available industrial robots can be classified into four categories according to their control systems.

The first category consists of limited-sequence robots which do not use a servo-control to indicate relative positions of the joints. Instead, they are controlled by setting limit switches and/or mechanical stops to establish the end

points of travel for each movement rather than by robot programming in the usual sense of the term.

The next category consists of playback robots with point-to-point control which use a more sophisticated control unit. A series of positions or motions are "taught" to the robot, recorded into memory, and then repeated by the robot under its own control. They are capable of performing motion cycles that consist of movement through a series of desired point locations and related actions. The robot is taught each point, and these points are recorded into the robot's control unit. Point-to-point robots do not control the path taken by the robot to get from one point to the other.

In the third category we have playback robots with continuous path control which are taught the positions or motions. They are capable of performing motion cycles in which the path followed by the robot is controlled. This is usually accomplished by making the robot move through a series of closely spaced points which describe the desired path. The individual points are defined by the control unit rather than the programmer. Straight line motion is a common form of continuous path control.

Finally, the fourth category includes intelligent robots which possess the capability not only to play back a programmed motion cycle but also to interact with the environment in a way that seems intelligent. Intelligent robots can alter their programmed cycle in response to conditions that occur in the workplace.

2.3.1 Basic Control Systems

A robot is a mechanical system that must be controlled in order to accomplish a useful task. The task involves the movement of the manipulator arm, and so the primary function of the robot control system is to position and orient the wrist and end effector with a specified speed and precision. When studying a mechanical system, we are concerned with the response of the

system to certain inputs. These inputs include commands to drive the system, and disturbances from the environment. The system is divided into five major components:

- (i) The input (or inputs) to the system,
- (ii) the controller and actuating devices,
- (iii) the plant (the mechanism or process being controlled),
- (iv) the output (the controlled variable), and
- (v) feedback elements (sensors).

Figure 2.1 Basic Control System.

By analyzing the effects that the controller and the plant have on the inputs, we can predict what the outputs will be under certain conditions. Figure 2.1 illustrates a general block diagram of the control system components for one joint of a robot manipulator. The input command is the position (and possibly the input speed) to which the joint is directed to move. The output variable is the actual position (and speed) of the joint. In nearly all modern-day robots,

most of the computational functions of the joint controller are carried out by a microprocessor.

Position and velocity sensors are used in robotics as feedback devices, while actuators (which provide the actual motive force) and power transmission devices are used to accomplish the control actions indicated by the controller. The purpose of the controller is to compare the actual output of the plant with the input command and to provide a control signal which will reduce the error to zero or to a quantity as close to zero as possible.

2.4 Compliance

The compliance of the robot manipulator refers to the displacement of the wrist end in response to a force or torque exerted against it. High compliance means that the wrist is displaced a large amount by a relatively small force. The term "springy" is sometimes used to describe a robot with high compliance. Low compliance means that the manipulator is relatively stiff and is not displaced easily. The study of compliance is important because it informs the user of the precision of movement of the robot when it is under load.

2.5 Kinematics

Kinematics is the branch of physics dealing with the concepts of motion without considering the particular forces or the masses of the bodies involved. In the case of robotics, kinematics relates the motion of a robot solely to the geometry of the system, without considering the masses of the robot arm or of its load, or the forces exerted on the components of the system either externally or by other parts of the system. There are two forms of robot kinematics, which are complements of each other: forward or direct kinematics, and reverse or inverse kinematics. Forward or direct kinematics produce a transformation from

the joint space to the Cartesian space. Reverse or inverse kinematics determine the inverse transformation of the forward kinematics, namely a transformation from the Cartesian space to the joint space.

To illustrate some of these concepts, we describe briefly a robot designed by Horn [102], and shown schematically in Figure 2.2. The robot has two rotary joints, whose positions are defined by the joint angles θ_1 and θ_2 . The joint axes are both parallel to the Z-axis, so that the robot moves in the X-Y plane. We assume that gravity acts in the negative "Y" direction. The lengths of the robot's links are L_1 and L_2 . Both are assumed to be cylinders of radius R , but schematically are drawn as straight lines.

2.5.1 Direct Kinematics

In this case, the position of the tip of the arm may be represented in a number of ways. One way is to utilize the two joint angles θ_1 and θ_2 , and this is known as the representation in joint space. Another way to define the arm position involves the use of a Cartesian coordinate system that may be external to the robot. The origin of the Cartesian axis system is often located in the robot's base, and the end of arm position would be defined in the Cartesian space.

Note that $L_1 \cos \theta_1$ is the length of the projection of P_1 on the X-axis, while $L_1 \sin \theta_1$ is the length of the projection of P_1 on the Y-axis. Hence, the position of the end effector is related to the joint angles by the following equation which is called the Forward Kinematics of Position of the robot.

$$P_2 = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2) \\ L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2) \end{bmatrix}$$

Figure 2.2 Direct or Forward Kinematics of Horn's 2-DOF Robot.

Thus, given θ_1 and θ_2 , it is always possible to compute the position of the end effector. The velocity of the end effector can also be determined by differentiating the above system with respect to time, if the angular velocities θ_1' and θ_2' are known. θ' denotes the angular velocity of a joint, and x' and y' are the horizontal and vertical components of the velocity of the end effector. Thus the matrix representation of the velocity of the end effector is:

$$P'_2 = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -L_1 \cos(\theta_1) - L_2 \sin(\theta_1 + \theta_2) & -L_2 \sin(\theta_1 + \theta_2) \\ L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2) & L_2 \cos(\theta_1 + \theta_2) \end{bmatrix} \begin{bmatrix} \theta'_1 \\ \theta'_2 \end{bmatrix}$$

or

$$P'_2 = J \cdot \theta'$$

Where:

$$J = \begin{bmatrix} -L_1 \cos(\theta_1) - L_2 \sin(\theta_1 + \theta_2) & -L_2 \sin(\theta_1 + \theta_2) \\ L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2) & L_2 \cos(\theta_1 + \theta_2) \end{bmatrix}$$

J is called the **Jacobian Matrix** of the robot.

Observe that the velocity, x' , y' can be obtained in terms of the angular velocity of the joints, $[\theta'_1, \theta'_2]$. Notice also that the velocity is a function not only of the angular velocity, but also of the position of the joints, (θ_1, θ_2) .

In a similar way, equations for the linear acceleration of the end effector can be derived by differentiating the above system.

2.5.2 Reverse Kinematics

Reverse or inverse kinematics give the joint space coordinates (joint angles) in terms of the Cartesian space coordinates. We shall present them for the Horn's robot.

The value of θ_1 and θ_2 can be determined in terms of x and y as follows. Consider Figure 2.3. Clearly,

$$\tan(\alpha + \theta_1) = y/x. \text{ Thus, } \theta_1 = \tan^{-1}(y/x) - \alpha$$

Figure 2.3 Inverse or Reverse Kinematics of Horn's 2-DOF Robot.

Now, by the Cosine Rule:

$$\cos \alpha = [L_1^2 + (\sqrt{x^2 + y^2})^2 - L_2^2] / [2L_1 \sqrt{x^2 + y^2}]. \text{ Thus}$$

$$\alpha = \cos^{-1}([L_1^2 + x^2 + y^2 - L_2^2] / [2L_1 \sqrt{x^2 + y^2}]), \text{ yielding,}$$

$$\theta_1 = \tan^{-1}(y/x) - \cos^{-1}([L_1^2 + x^2 + y^2 - L_2^2] / [2L_1 \sqrt{x^2 + y^2}])$$

Using these, the value of θ_2 can also be determined in terms of x and y using the Cosine Rule on the angle between L_1 and L_2 , yielding,

$$\cos(180 - \theta_2) = [L_1^2 + L_2^2 - (\sqrt{x^2 + y^2})^2] / [2L_1 L_2]$$

hence,

$$\theta_2 = \cos^{-1}\{[x^2 + y^2 - L_1^2 - L_2^2] / 2L_1 L_2\}$$

In general, there are two solutions to these equations, because they involve both $\cos^{-1}$ and $\tan^{-1}$. In some robot arms with six degrees of freedom,

there are eight different solutions to the reverse kinematics. At certain points, the solution is degenerate. For instance, if (x,y) is the origin, then $\tan^{-1}(y/x)$ is indeterminate. Also, notice that if the point (x,y) is outside of the workspace, there is no solution to the inverse kinematics.

A few remarks are not out of place:

- (i) If $\theta_2 = 0$ (J does not have an inverse), then a pseudo inverse must be used.
- (ii) J may not be a square matrix for certain types of robot arms. In this case the pseudo inverse $(JJ^T)^{-1}$ must be used.
- (iii) For the robot arm with six degrees of freedom, J is a square matrix, but inverting it is computationally very expensive.
- (iv) When the robot is near a singularity (e.g. when $\sin\theta_2 = 0$), inversion of the matrices will become ill-conditioned. This may occur, for instance, if a joint is almost straight (0 degrees) or if a joint is almost doubled back on itself (180 degrees).

The manipulator previously described is a very simple; a two-jointed robot arm has little practical value except for very simple tasks. If we add to the manipulator the modest capability of orienting itself as well as the ability to position its end effector, we will have to incorporate a third degree of freedom as shown in Figure 2.4. In this case the forward kinematics become:

$$x = L_1 \cos \theta_1 + L_2 \cos (\theta_1 + \theta_2) + L_3 \cos (\theta_1 + \theta_2 + \theta_3)$$

$$y = L_1 \sin \theta_1 + L_2 \sin (\theta_1 + \theta_2) + L_3 \sin (\theta_1 + \theta_2 + \theta_3)$$

$$\alpha = (\theta_1 + \theta_2 + \theta_3)$$

Figure 2.4 Forward Kinematics of Horn's 3-DOF Robot.

2.6 Dynamics

The topic of robot dynamics is concerned with the analysis of the torques and forces due to acceleration and deceleration. Torques experienced by the joints due to acceleration of the links, as well as forces experienced by the links due to torques applied by the joints, are included within the scope of dynamic analysis. Solving for the acceleration is dependent on the inertia of the arm. However, the inertia is dependent on the configuration of the arm, and this is continually changing as the joints are moved. An additional factor that influences the inertia is the mass of the pay load and its position with respect to the joints. This also changes as the joints are moved.

The torques required to drive the robot arm are not only determined by the static and dynamic forces; each joint must also react to the torques of other

joints in the manipulator, and the effects of these reactions must be included in the analysis. Also, if one joint of the arm rotates relative to another, the centrifugal effect will have to be considered. We shall not study the dynamical properties of any robot in this thesis.

2.7 Sensors

Sensors, used as peripheral devices in robotics, include both simple types such as limit switches and sophisticated types such as machine vision systems. Of course, sensors are also used as integral components of the robot's position feedback control system. Their function as peripheral devices in a robotic work cell is to permit the robot's activities to be coordinated with other activities in the cell. The sensors used in robotics include the following general categories:

Tactile sensors, which respond to contact forces with another object. Some of these devices are capable of measuring the level of force involved.

Proximity sensors, that indicate when an object is close to another object before a physical contact has been made. When the distance between the objects can be measured, the device is called a range sensor.

Miscellaneous sensors, encompassing the remaining categories of sensors used in robotics which include sensors used for temperature, pressure, etc.

Machine vision systems, capable of viewing the work space and interpreting what is seen. These systems are used in robotics to perform inspection, and part recognition.

2.8 Precision of Movement

Precision of Movement is a measure of robot performance. Precision can

be described with three parameters; spatial resolution, accuracy and repeatability.

The spatial resolution of a robot is the smallest increment of movement into which a robot can divide a work volume. It depends on two factors, namely the system control resolution and the mechanical inaccuracies of the robot.

Accuracy refers to the ability of a robot to position the end effector at a desired target point within the work volume. The accuracy of a robot can be defined in terms of its spatial resolution because the ability to achieve a given target point depends on how closely the robot can define the control increments for each joint motion.

Repeatability is concerned with the ability of a robot to position the wrist or end effector at a point in space that had previously been taught to the robot. Repeatability and accuracy refer to two different aspects of the robot's precision. Accuracy relates to the capacity of a robot to be programmed to achieve a given target point while repeatability refers to the ability to return to the programmed point when commanded to do so. These concepts are illustrated in Figure 2.4.[35].

Repeatability errors form a random variable and constitute a probabilistic distribution as shown in Figure 2.4. It would be convenient if the repeatability error formed a bell-shaped curve, suggesting a normally distributed random variable. In reality, the mechanical inaccuracies that are chiefly responsible for repeated errors in each joint do not form the symmetric distribution shown in Figure 2.4. However, when the errors from several axes of motion are combined, the resulting aggregate error is influenced by the central limit theorem in probability, which states that the sums of random variables tend to form a normally distributed variable, even though the individual random variables are not normally distributed. Accordingly, it can be inferred that the repeatability error of a robot with five or six axes is approximately normal, even

Figure 2.5 Repeatability and Accuracy.

if the error due to each individual axis is not normal.

2.8.1 Position Errors

Although the desired position and orientation are communicated to the robot processor through teaching and/or other methods, the actual position and orientation of the hand will often be somewhat different from those desired. The main causes of this error have been classified as errors in joint variables (joint angles and/or displacements), and errors in link parameters (length, twist, distance). Two methods for calculation of hand errors as a function of errors in kinematic parameters are given in Wu [104] and Hayati [37].

In the absence of link parameter errors, joint position errors will be the controlling factor in the precision operation of the robot. The joint position error

is defined as the difference between the angles/displacements actually assumed by a joint and the commanded values. These errors are the result of several factors including the dynamic effects, friction, and random unaccountable phenomena.

These errors can be decomposed into two components. One component is a deterministic error which always acts in the same direction. For instance, the dynamic effects may be such that, for a given operation, a particular joint angle would always be larger than its desired value. This portion of the error is called the bias of the joint position error. The other portion of the induced error, which is the stochastic part, is due to random or unknown factors whose effects do not have a particular direction. We refer to this component as the random error [14].

One must note that although the bias in one joint position depends on the other joints through the complicated dynamic and kinematic relations governing the robot motion, random errors in the joints could be independent of each other.

2.9 Conclusion

In this chapter we have briefly reviewed the concepts that will be utilized throughout the succeeding chapters. These concepts include the anatomy of a robot, its motion characteristics, its control systems, its kinematics and dynamics, and have described some of the latter using a simple robot called Horn's Robot. We have also briefly remarked on the position errors encountered by robot manipulator arms.

Chapter 3

Previous Methods

As we noted earlier, a basic problem in Robotics is that of planning motions to solve some previously specified task and then controlling the robot with the primitive commands necessary to execute those motions. Trajectory planning converts a description of desired motion to a trajectory, defining the time sequence of intermediate configurations of the arm between the source and the destination positions.

In this chapter we review some of the previous algorithms in trajectory planning. In the first part of chapter three, we outline a brief history of trajectory planning algorithms, explaining in detail the one using "On-line Splines" for the Joint Trajectory Interpolation algorithm [23]. A modified version of the latter is presented, in which noise has been added to each joint of the link. In the second part of the chapter we give a brief overview of control algorithms, explaining in detail a learning method due to Thomas Miller III, which is akin to adaptive control approaches [61]. Finally we discuss the "Bettering Process", due to Arimoto *et. al* [10].

3.1 Brief History on Trajectory Planning

Trajectory planning converts a description of a desired motion to a trajectory defining the time sequence of intermediate configurations of the arm between the source P_0 and the destination P_1 . As the trajectory is executed, the tip of the end effector traces a curve in space and changes its orientation. The net effect is a translation and a rotation of the end effector. As noted earlier, the space curve traced by the end effector is called the trajectory.

Mathematically, a path is a function of a single parameter T , which signifies the proportion of the curve that has been traversed. The parameter

can be normalized so that $T = 0$ represents the source configuration P_0 , and $T = 1$ represents the destination P_1 . Analogously, the time sequence of orientations of the end effector traces a curve in a three dimensional space of orientations, say Euler angles (see Paul [80] page 43). Mathematically however, a rotation curve is once more a function of a single parameter. Although rotations and translations can be combined into a single six dimensional space, it is simpler to treat them separately. The path and rotation curve are single-parameter functions that correspond to a configuration. In general, a trajectory includes information about velocity and acceleration of the end effector. If the parameter is identified with normalized time, differentiating the latter yields the translational and angular velocity and acceleration of the arm. The output of a trajectory planning module is a sequence of arm configurations that forms the input to the arm's feedback control system.

Two major algorithms in trajectory planning are the one for straight line motion in Cartesian space, due to Taylor [89] and the one involving polynomial trajectories in joint space by Paul [80, 81]. These two algorithms are well known in the literature and have been the underlying basis for others. In this thesis we will not explain in detail any of these algorithms.

Taylor proposed two methods for achieving straight line motions [89]. The first, "Cartesian Path Control", is a refinement of a method developed by Paul [81]. Intermediate points are interpolated along the Cartesian straight line path at regular intervals during the motion, and the manipulator's kinematic equations are solved to produce the corresponding intermediate joint parameter values. Taylor's second method, called the "Bounded Deviation Joint" method, uses a motion planning phase to precompute enough intermediate points so that the manipulator may be driven by interpolation of joint parameter values while keeping the tool on an approximately straight line path. This technique allows a substantial reduction in real time computation

and permits problems arising from degenerate joint alignments to be handled more easily. The planning is done by an efficient recursive algorithm which generates only enough intermediate points to guarantee that the tool's deviation from a straight line path stays within prespecified error bounds.

One approach to generate robot trajectories is to use polynomial trajectories in joint space (Paul [81], Derby [27]). Thus, the displacement for each coordinate for a 3-4-5 polynomial trajectory is,

$$q = q_i + (q_f - q_i) [10 (t/T)^3 - 15 (t/T)^4 + 6 (t/T)^5]$$

where $(q_f - q_i)$ is the total coordinate displacement in a path segment and T is the time period for the traversal of the entire path segment.

The above 3-4-5 polynomial constrains both first and second derivatives to be zero at path-segment endpoints. In order to constrain derivatives to arbitrary values at path segment endpoints, a more general polynomial must be used. Derivatives up to order M can be constrained at path endpoints using a general polynomial of order $2M+1$ for the trajectory. Numerical determination of the coefficients for an M th-order polynomial involves solving a set of $M+1$ simultaneous linear equations, which become difficult to solve accurately as the order of the polynomial increases. To circumvent this problem, Hornick and Ravani [43], propose the use of truncated trigonometric series as a basis for generating robot trajectories. The trigonometric displacement can be written as:

$$q = q_i + (q_f - q_i) (t/T) + \sum_{1 \leq n \leq N} [a_n T / n^2 \pi^2 \sin(n\pi t / T)]$$

The use of truncated trigonometric series allows the endpoint derivatives to be matched with only very few terms in the series. Actually, all even derivatives at endpoints are implicitly set to zero because of the sinusoidal form

of the series. The series coefficients, a_n , are determined by specifying the appropriate boundary conditions at the endpoints of a path segment.

In the robot trajectory planning, the proper choice of the time period of evaluation, T , of the motion (cycle time) is important. Hollerbach [42] has suggested a method for time-rescaling, and his techniques have been extended to include trajectory profiles with multiple segments, by Hornick and Ravani [43].

The use of polynomial functions for generating smooth robot trajectories was studied more by Lewis [48], Finkel [29], and Luh *et. al* [54]. It has been shown that many undesirable properties of high-order polynomials can be overcome by using lower-order spline functions (Finkel [29], Bollinger and Duffie [17]). If a trajectory has a large number of points, then a spline fit for the trajectory requires all spline coefficients to be computed before the trajectory is executed because of the inherent nature of spline calculations (Ahlberg *et. al* [1], DeBoor [26]). Hence a spline trajectory with a large number of intermediate points has to be computed off-line. Techniques for piecewise spline paths using X-splines have also appeared (Clenshaw *et. al* [21]; Luh *et. al* [54]). Obtaining the ideal look-ahead points for a piecewise cubic spline trajectory and the techniques for breaking the off-line spline computations into smaller blocks for real-time trajectory calculations have been investigated by Chand and Doty [23]. This algorithm will be explained in the following section.

3.2 On-line Cubic Splines for Joint Trajectory Interpolation

An end effector move is executed by transforming its position and orientation into joint coordinates at a large number of points on the proposed trajectory. If the move is to be in real time with an on-line trajectory calculation, the robot controller is allowed to look ahead at only a limited number of points along the trajectory. The scheme developed by Chand and Doty [23] involves

estimating an additional constraint at the last point of a look-ahead block and building cubic spline trajectories for interpolating the points within the look-ahead block.

Figure 3.1 Sample Points on a Joint Trajectory.

A segment of a joint trajectory with the starting point P_1 and two intermediate points P_2 and P_3 is shown in the Figure 3.1. The block size is assumed to be three (i.e., each block has two look-ahead points). One way of determining an additional constraint at the last point of a look-ahead block is to compute the joint velocities by using the inverse kinematics of velocity. Two general approaches to obtain this are the use of the inverse Jacobian matrix, Whitney [103] and the application of screw theory, Lipkin and Duffy [49]. The data required for the inverse kinematics of velocity are the linear and rotational velocities of the end effector and the corresponding configuration of the arm. The complexity of the inverse velocity transformation is dependent on the geometry of the manipulator. In general, it is difficult to solve for the inverse kinematic solution, but the screw theory suggests a simplified method. The study of the screw theory is not the primary aim of this thesis.

With position and velocity constraints at the start and end points of a block, velocities at each intermediate point of the block can be estimated by imposing continuity constraints in acceleration and solving a tridiagonal matrix equation, DeBoor [26], Luh *et. al* [54]. The tridiagonal matrix equation is as follows:

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & \dots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & \dots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \dots & 0 \\ & & & & & & \\ & & & & & & \\ 0 & 0 & 0 & 0 & 0 & \dots & a_{n,n-1}a_{n,n} \end{bmatrix} \begin{bmatrix} v(1) \\ v(2) \\ v(3) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ (b_n - v_{n+1})T_n \end{bmatrix}$$

where the coefficients a_{ij} and b_k are

$$a_{i,i} = 2(T_i + T_{i+1}),$$

$$a_{i,i-1} = T_{i+1},$$

$$a_{i,i+1} = T_i,$$

$$b_i = (3 \cdot T_{i+1} / T_i) (\theta_i - \theta_{i-1}) + (3 \cdot T_i / T_{i+1}) (\theta_{i+1} - \theta_i),$$

with T_i, T_{i+1} being the i th and the $(i+1)$ st time intervals, θ_i the i th joint position at time t_i , and v_0 and v_{n+1} the velocities at the start and destination points.

A recursive solution to the matrix equation can be derived as follows. Define

$$\gamma_j = a_{j,j+1} / \beta_j, \quad j = 1, 2, \dots, n-1,$$

$$\gamma_n = T_n / \beta_n,$$

$$\beta_j = a_{j,j} - a_{j,j-1} \gamma_{j-1}, \quad j = 2, \dots, n,$$

$$\beta_1 = a_{11}.$$

Also define

$$z_j = (b_j - a_{j,j-1}) / \beta_j, \quad j = 2, \dots, n,$$

$$z_1 = (b_1 - v_0 T_1) / \beta_1.$$

Then,

$$v_j = z_j - v_{j+1} \gamma_j, \quad j = n, \dots, 1.$$

The equations so derived will demonstrate the effect of a change or error in the velocity v_{n+1} on the intermediate velocities. Thus, v_{n+1} becomes

$$v_{n+1} + \Delta v_{n+1},$$

$$\text{where } \Delta v_j = u_j (\Delta v_{n+1})$$

$$\text{and } u_j = (T_j / \beta_j) (T_{j+1} / \beta_{j+1}) \dots (T_n / \beta_n) = \prod_{k=j}^n \gamma_k$$

Thus, with position and velocity constraints at each intermediate point of the joint trajectory and ignoring the acceleration at the start and end points, a cubic polynomial can be derived for the entire trajectory. The first and last polynomials will be quartics if the initial and final point accelerations are considered. The interpolating curve will be continuous in the first and second derivatives at all points except the block endpoints where continuity in the second derivatives is lost.

We now outline the equations for generating the spline coefficients for each block assuming a two look-ahead strategy. A normalized time "u" for each segment of the joint trajectory is defined as follows. For the i^{th} segment, let

$$u = (t - t_{i-1}) / T_{i-1},$$

where

$$T_{i-1} = t_i - t_{i-1}.$$

All polynomials are expressed with respect to u , where on time segment i

$$du / dt = 1 / T_{i-1}$$

Let

$$f_i(u) = a_{3i} u^3 + a_{2i} u^2 + a_{1i} u + a_{0i}$$

For the i th segment, $f_i(u)$ is evaluated from $u = 0$ to $u = 1$ as follows:

$$\begin{aligned} df_i(u) / du &= (df_i(u) / dt) \cdot (du / dt) \\ &= T_{i-1} (df_i(u) / dt) \end{aligned}$$

Also,

$$d^2(f_i(u)) / du^2 = T_{i-1}^2 (d^2 f_i(u) / dt^2)$$

The last two equations transform the velocity and acceleration of the joint at t_i to velocity and acceleration of the normalized trajectory. Let

$$f_1(u) = a_{31} u^3 + a_{21} u^2 + a_{11} u + a_{01} = 0, \quad (3.1)$$

$$f_2(u) = a_{32} u^3 + a_{22} u^2 + a_{12} u + a_{02} = 0, \quad (3.2)$$

be the cubic polynomials describing the trajectory between P_1 and P_2 , and between P_2 and P_3 , respectively. Also, let θ_1 and θ_3 denote the velocities at P_1 and P_3 respectively, where, θ_1 is known from the polynomial preceding P_1 . If P_1 is the start point of the trajectory and the arm starts from rest, then $\theta_1 = 0$. θ_3 is determined by the reverse kinematics of velocity at time instant t_3 (point P_3). Splining the last two equations together with the continuity constraints in acceleration at P_2 , yields an expression for θ_2 as follows:

$$(6a_{31}u + 2a_{21}) / T_{21}^2 \big|_{u=1} = (6a_{32}u + 2a_{22}) / T_{22}^2 \big|_{u=0} \quad (3.3)$$

Expressing all coefficients in the equations (3.1) and (3.2) in terms of θ_1 , θ_2 , θ_3 , θ'_1 , θ'_2 , and θ'_3 , they become:

$$a_{01} = \theta_1, \quad a_{11} = \theta'_1 T_1,$$

$$a_{21} = 3(\theta_2 - \theta_1) - (\theta'_2 + 2\theta'_1) T_1,$$

$$a_{31} = -2(\theta_2 - \theta_1) + (\theta'_1 + \theta'_2) T_1,$$

$$a_{02} = \theta_2, \quad a_{12} = \theta'_2 T_2,$$

$$a_{22} = 3(\theta_3 - \theta_2) - (\theta'_3 + 2\theta'_2) T_2,$$

$$a_{32} = -2(\theta_3 - \theta_2) + (\theta'_2 + \theta'_3) T_2.$$

Substituting in equation (3.3), and solving for θ'_2 , we have

$$\theta'_2 = 1/2 (T_1 + T_2) [(3T_2 / T_1) (\theta_2 - \theta_1) + (3T_1 / T_2) (\theta_3 - \theta_2) - \theta'_1 T_2 - \theta'_3 T_1].$$

An advantage of using actual joint velocities as constraints at certain intermediate points is that each joint will closely track the actual joint velocities that are required to maintain the desired end effector velocity. This implies that the velocity profile of the end effector between intermediate points and at initial and final points will be closer to the desired value.

3.2.1 Experimental Results

The algorithm described above was implemented, using a two-point look-ahead strategy. Also, a modified version of the algorithm was implemented to incorporate the possibility of the final position being noisy. In each case, one hundred experiments were performed and the expected path of

the robot end effector was traced. The expected trajectory was calculated after one hundred iterations.

Various simulations were done for a variety of start and end points. We present a **typical** example of a simulation for the Horn's Robot for the motion described below:

Initial Joint Angle $\theta_1 = 30^\circ$,

Initial Joint Angle $\theta_2 = 60^\circ$,

Final Joint Angle $\theta_1 = 60^\circ$,

Final Joint Angle $\theta_2 = 40^\circ$.

The expected trajectory is shown in Figure 3.2a and the corresponding decrease in the Euclidian distance from the final configuration is shown in Figure 3.2b as a function of time.

Figure 3.2a Expected Trajectory of Horn's Robot using the Cubic Spline Interpolation technique described in Section 3.2. The details of the initial and final configurations are given in Section 3.2.1.

Expected
distance

Figure 3.2b Expected change in distance for the scheme described in Section 3.2 from the initial configuration to the final mean configuration. The set-up is described in Figure 3.2a.

3.3. Brief History of Control Algorithms

One important aspect of gripper and manipulator design is in the area of their control, which can be divided into two categories: the first deals with the actual physical control (electronic, pneumatic, hydraulic, and mechanical aspects) and the second deals with the mathematical theory of control systems applied to the design and programming of robotic manipulators. In this thesis we are concerned with the second group.

Mathematical algorithms were developed for predicting and generating the motion of manipulators. One, by Loeff and Soni [52], generated incremental motions for each position of the manipulator. A recursive routine using this algorithm would automatically direct the manipulator to its specified destination while avoiding collisions with obstacles in its path.

Mathematical models have also been developed to represent the motion of the robot. Birk [16] suggested a set of empirical equations that enabled a robot having six degrees of freedom to orient and position workpieces. The following constraints applied to his solution:

- (i) The robot can select a single part from a random collection of similar parts without a vision system.
- (ii) There are three points from which the spatial locations of both the part and the fixture can be determined.

A manipulator working under Birk's algorithm would learn to place the part in its optimal position. Taking a different point of view, Murata [66] developed an algorithm for the real-time control of manipulators based on the kinematics (rather than the dynamics) of the arm. Since this algorithm was based on kinematics, it no longer required the iterative solution of differential and algebraic equations. This in turn allowed for faster and easier programming and a shorter run time for the processor that was controlling the robot.

Taking yet a different approach, Luh and Walker [55] used linear programming methods to solve minimum travel time problems for manipulators. They based their method on the assumptions that the acceleration and velocity constraints were known and that the minimum positional error constraints can be determined (or assumed) based on the configuration of the manipulator.

As another approach to mechanical arm control, Raibert [82] used a state space representation in which this system is implicitly able to control the manipulator, to adapt to mechanical changes and to perform coordinate transformations of the manipulator. This in turn generates a table of equations of motion that are governed by the state variables, and these equations are implemented using the data derived from exploratory previous motions undertaken by the robot.

In addition to the determination of the robot's position in space and time, important considerations are the ability of the gripper to grasp parts in a stable manner, to offer feedback as to how much force is being applied, and, in some cases, to determine its proximity to the part in question. In a study of stable prehension of a robotic hand, Hanafusa and Asada [38] found that stable prehension is generated by using a stability condition which is returned to the robot from the gripper. Gerelle [32] used this concept and proposed a force feedback method of control to provide a tactile sensing mechanism, which he claimed was essential for the implementation of control in industrial robots. He suggested that a force sensor be used to provide information for force feedback, which in turn is used to modify the robot's movements.

In researching industrial manipulators, Stokic and Vukobratovic [85] found that the major driving force is the coupling between the various degrees of freedom. They also suggested that it would be advantageous if this force could be measured directly. They stated that once some sort of feedback could be generated with respect to this coupling, system stability could be enhanced, and

greater control could be achieved using this global force feedback.

Takegaki and Arimoto[11] also used a global control for gross motion to achieve higher levels of control than would otherwise be possible. By using linear feedback of generalized velocities and coordinates for these gross motions, they found that the process could be commanded by a task oriented control. One major drawback to this system is the extreme complexity inherent with the system, which requires either some form of coprocessing or an extremely fast microprocessor to cope with the vast amount of information being generated by the system.

Raibert and Craig [25], developed a hybrid technique of feedback control using position, orientation and a force-torque control. Using a force sensor at the wrist to measure force-torque with position sensors located at the manipulator joints, control is achieved through the feedback that the task and its parameters impose on the manipulator.

Hewitt [40] proposed a method for decoupling of nonlinear dynamic systems, such as those that are often present in robotic manipulators. Using his method, one obtains decoupled controls for the equations of motion for industrial robotic arms. In a later work he [41] used an alternate form of active force control to obtain decoupled motions in the arm. Thus, he was able to derive a control algorithm that required only partial system modeling and could be run in real time.

Togai and Yamano [91] proposed a discrete learning control algorithm based on state variable techniques for time invariant systems and obtained the condition for the convergence of the system state, which contains velocity as well as position, and thus extended the previous work done in the area of linear time varying dynamic systems. The advantage of the state variable approach is that it allows a similar representation of both continuous and discrete systems. The continuous system is represented by a set of first-order differential state

equations, and a discrete system by a set of first order difference equations.

Various adaptive control schemes based on interactive signal synthesis have also been published. The first paper on repeated trajectory learning seems to have been due to Uchiyama [96]. Subsequent works include that of Arimoto *et. al*, [4, 5, 6, 7, 10,12], Hara *et. al* [36], Craig [25], Mita and Kato [63], Togai and Yamano [91], Wang [100], and Wang and Horowitz [101]. These papers discuss the use of linear learning operators and have emphasized stability of the proposed algorithms. Fu [30] identified five learning algorithms mostly based on stochastic or statistical approaches, which have been widely used in pattern recognition and psychology. Arimoto and his colleagues [4, 5, 6, 7, 10, 12], and Craig [25] introduced some new concepts in learning control theory and they applied these algorithms to control the robots. Mita [63] discussed the stability of the learning algorithms in the frequency domain assuming that the process input is a periodic function. Since the process input is subject to change in the course of the learning operation, it is inadequate to assume that the input is periodic. Moreover, the frequency domain analysis is valid only for continuous systems. Since in real life, digital controllers are used both to control robot manipulators and to store the information obtained in the course of learning processes, a discrete system analysis is more appropriate than a continuous analysis.

There has been little work done to evaluate the performance, (i.e. the convergence rate) of the various algorithm. Simulations of several of these algorithms have revealed very slow convergence, large sensitivity to disturbances, and high sensitivity to sensor and actuator noise [24].

3.4 Miller's Algorithm

In this section we shall discuss in some detail a learning control approach due to Miller III [61]. It is based on the Cerebellar Model Articulation Controller

(CMAC) model described by Abus[3]. Miller adapted the above model to robotics. In his scheme the CMAC memory is trained to reproduce the relationship between available sensory feedback parameters and system command parameters over regions of the system state space, rather than being trained to generate the command signals required to follow particular trajectories. Training is performed at the end of each control cycle based on an on-line observation of the system input-output relationships and is independent of the immediate control goals. The learned information is then used at the beginning of each cycle to predict the commands required to produce desired changes in the feedback parameters. Information learned during previous movements can be utilized to achieve new control goals as long as it involves similar regions of the system state space.

3.4.1 The CMAC Module

The CMAC module is designed to represent, in a table lookup fashion, functions of the form

$$g(f(s))$$

where s is a multidimensional discrete input state vector, g is a multidimensional vector output, and f is an arbitrary function relating s to g .

In the CMAC module, each discrete input state s maps to many locations in the memory (one "location" contains a vector of the same dimension as g), and the function $f(s)$ is computed by summing the values at all of the locations mapped to s . The mapping algorithm has the characteristic that nearby states in the space s map to overlapping locations in the memory A , while distant states in the space map to almost "independent" locations in the memory. The amount of overlap between two sets of locations in A is related to the summed absolute difference between the corresponding discrete vectors in s .

The resulting system will produce an output $f(s)$ for any input vector s in S which represents the entire state space.

3.4.2 CMAC-Based Control

The general sensory-interactive control problem can be posed as one of determining the sequence of robot command signals required to perform a task, which is expressed in terms of a sequence of particular sensor outputs. At the beginning of each control cycle, a trajectory planner determines the sensor outputs desired at the end of the control cycle. This is done based on an ideal task description. These desired sensor values are combined with the sensor outputs defining the current control state to form the desired vector s_d .

The CMAC module output $f(s_d)$ is then computed and is used to command the robot. The sensor outputs defining the control state at the beginning of the control cycle are combined with the current outputs of the sensors to form the observed vector s_o . The CMAC module output $f(s_o)$ is then computed. Ideally $f(s_o)$ should be equal to the actual robot command applied during the control cycle g_o . Each of the memory locations mapped to by s_o is adjusted by adding the value

$$\delta = \beta * (g_o - f(s_o)) / C$$

where C is the total number of memory locations mapped by s_o and $\beta \in [0,1]$ represents a learning factor. If β is chosen to be unity, the memory locations are adjusted such that $f(s_o)$ is equal to g_o . On the other hand, if $\beta < 1$, the memory locations are adjusted such that the CMAC output $f(s_o)$ is changed in the direction of g_o .

Note that the information learned is independent of the control objective for the control cycle, and this information is potentially very useful, because it tells the user whether the system is learning, and whether or not the current control objectives are being satisfied.

In [61], Miller III used two different techniques to generate initial performances, and using these an initial learning phase was carried out. In the first experiment a series of random robot movements was used, and in the second experiment a fixed-gain linear feedback controller placed in parallel with the CMAC controller was used.

3.4.3 Static Positioning Experiment

At the start of the experiment, all of the A memory locations contained null vectors such that $f(s_d)$ would be a null command for all input states s_d , and learning would not occur without some initialization procedure. In the learning controller, the discrete state vector s was considered to be composed of the positions of the three robot axes at the beginning of a control cycle (the control state) and the unit vector components s_x , s_y , and s_z specifying the direction of robot motion during the control cycle.

At the beginning of each control cycle, the values of s_x , s_y , and s_z from the current video image and the current joint positions were determined and used to form the discrete CMAC input state vector s_d . The CMAC output $f(s_d)$ was then computed and scaled by a distance factor d_{xyz} (in pixel-units) to form the joint position change command sent to the robot. The control cycle ended when the commanded movement was completed. The evaluation of the movement of objects in the image field could be simplified if large movements were avoided. Thus, before a command was sent to the robot, the summed absolute change in motor position was computed. In Miller's experiments, if

this value was greater than eight degrees, the three motor command values were scaled proportionally such that the summed absolute change in angle was eight degrees.

3.4.4 The Conveyor Tracking Experiment

As in the above experiment, the CMAC memory was initialized to a null state, so that $f(s_d)$ would produce a zero velocity command for any s_d and an initialization procedure was invoked. A fixed-gain linear feedback controller was implemented in parallel to the learning controller to provide initial movements upon which the controller could learn and improve. A qualitative examination of the orientation of the conveyor relative to the robot suggested the following control structure

$$\theta'_1 = k * d_x$$

$$\theta'_2 = -k * d_y$$

$$\theta'_3 = -k * d_z$$

where d_x , d_y , and d_z are the changes in disk coordinates required to center the disk in the image field, k is a control gain, and θ'_1 , θ'_2 , and θ'_3 correspond to the commanded velocities of the base, upper arm and forearm axes of the robot respectively.

In both experiments, the control system performance was found to be limited by the resolution of the sensor feedback data, rather than by control structure limitations.

An important feature of the CMAC-based learning control technique proposed is that, similar to adaptive control approaches, the resulting control software contains only qualitative information about the system being controlled. In some applications, however, the learning control approach has a

major disadvantage. With a traditional controller, the control system performance can sometimes be predicted analytically whereas the learning controller can only be evaluated by experiment.

3.5 Betterment Process

In this section we shall describe a technique called Betterment due to Arimoto *et. al* [5, 8, 9, 12]. It is based on a simple iterative learning algorithm and it updates the control input based on the previous operation data and improves the performance of next operation (in a certain sense), provided a desired motion trajectory is given.

Arimoto *et. al* [10], proposed three types of learning control laws for mechanical or mechatronic systems with linear and nonlinear dynamics. Given a desired output y_d over a finite time duration $[0, T]$ and an appropriate input u_0 , these laws are formed by the following simple iterative processes

$$\begin{aligned} \text{(i)} \quad u_{k+1} &= u_k + \phi (y_d - y_k) \\ \text{(ii)} \quad u_{k+1} &= u_k + \Gamma d/dt (y_d - y_k) \\ \text{(iii)} \quad u_{k+1} &= y_k + (\phi + \Gamma d/dt) (y_d - y_k) \end{aligned} \quad (3.4)$$

In the above, u_k (u_{k+1}) denotes the k^{th} ($k+1$)st input, y_k the measured output at the k^{th} instant corresponding to u_k , and ϕ and Γ are positive definite constant gain matrices. The first law with an appropriate gain matrix ϕ is convergent in the sense that $y_k(t)$ approaches $y_d(t)$ as $k \rightarrow \infty$.

The resultant learning control method can be described by a simple iterative rule:

$$u_{k+1}(t) = u_k(t) + \Gamma d/dt \{y_d(t) - y_k(t)\}, \quad (3.5)$$

where Γ is an $m \times m$ constant matrix called the "gain matrix". Informally, it can

be seen that in the $(k+1)^{\text{st}}$ run, the control input is updated in such a way that the previous input at the k^{th} run is modified by a quantity related to the derivative of the error between the previous output $y_k(t)$ and the given desired output $y_d(t)$.

Suppose that a finite-dimensional dynamic system has a control input vector u of dimension m and a measured output vector y also of dimension m . Then, given a desired output $y_d(t)$ over a finite time duration $[0, T]$, it is required to find a control input $u^*(t)$, which excites the system and eventually produces an output $y^*(t)$ so that $y^*(t)$ must be coincident with $y_d(t)$ over $t \in [0, T]$. In practical situations, when it is usual that only a rough sketch of the system is available, such a Betterment Process is very effective.

The three iterative processes defined in (3.4) are classified as follows:

- (i) C^0 - Type Betterment Process

$$u_{k+1}(t) = u_k(t) + \phi e_k(t),$$

where

$$e_k = y_d(t) - y_k(t)$$

and ϕ is an $m \times m$ constant gain matrix.

- (ii) C^1 - Type Betterment Process, which is schematically shown in Figure 3.3.

$$u_{k+1}(t) = u_k(t) + \Gamma \frac{d}{dt} e_k(t)$$

- (iii) Mixed -Type Betterment Process

$$u_{k+1}(t) = y_k(t) + (\phi + \Gamma \frac{d}{dt}) e_k(t).$$

There are two approaches of the Betterment Process methods that apply

to robot manipulators. It is well known that the dynamics of serial-link manipulators with n degrees of freedom is described by

$$(J_0 + H(q)) \ddot{q} + f(q, \dot{q}) + g(q) = Ku, \quad (3.6)$$

where $q \in R^n$ is the vector of joint coordinates, H is a positive definite inertial matrix, J_0 is a positive diagonal matrix representing the inertial terms of internal load distribution of the actuators, $f(q, \dot{q})$ is a vector-valued function of centrifugal, Coriolis, and viscous frictional forces, $g(p)$ is a vector due to the gravity force, u is a vector of input voltages given to actuator servomotors, and

Figure 3.3 C¹ - Type Betterment Process.

K is the diagonal gain matrix.

This nonlinear system representation may be rewritten into the state space form as below:

$$\dot{q} = p,$$

$$\dot{p} = -(J_0 + H(q))^{-1} [f(q, p) + g(q)] + (J_0 + H(q))^{-1} Ku$$

It is assumed that velocity vector $p = \dot{q}$ can be measured and thereby regarded as an output. Let $x^t = [q^t, p^t]$. Then the robot system may be expressed as:

$$\dot{x} = f(x) + B(x) u,$$

$$y = [0, I] x = Cx.$$

For a desired velocity profile $y_d = q_d(t)$, given over $t \in [0, T]$, the C^1 -Type Betterment Process converges if an initial input $u_0(t)$ is adequately chosen so that the corresponding output $y_0(t)$ lies in the neighbourhood of $y_d(t)$ and, in addition, the following condition is satisfied for all q

$$\|I - (J_0 + H(q))^{-1} K \Gamma\| < 1.$$

In the case that a desired path is given and described in terms of task coordinates $y = (y_1, \dots, y_m)$, the proposed learning scheme must be modified as follows:

$$u_{k+1}(t) = u(t) + J^T(q(t)) \Gamma \frac{d}{dt} e_k(t)$$

The use of C^1 -Type Betterment Process for the system means that differentiation of the velocity vector must be carried out.

In practice, the velocity signal measured through a tachogenerator is often contaminated by noise. Hence, if numerical differentiation does not work well in such a case, the use of C^0 -Type Betterment Process can be recommended on the basis of the following argument.

First of all, we note that given a desired trajectory $y'_d = \dot{q}_d$ on $[0, T]$, equation (3.6) can be rewritten as a linear differential equation in terms of deviation vectors $x'_k = y_k - y_d$ and $v_k = u_k - u_d$ as in (3.7):

$$(J_0 + H(t)) x''_k + B(t) x'_k + A(t) x_k = o(x_k) + K v_k(t) \quad (3.7)$$

In the above, u_d represents an unknown desired input that realizes the desired output $y_d(t)$, $H(t) = H(q_d(t))$, $B(t)$ and $A(t)$ are appropriate matrix-valued functions which are dependent on the choice of $q_d(t)$, and the symbol

$o(x)$ means a term of an order higher than x , x' , and x'' .

By definition, the inertial matrix $J_0 + H(t)$ is positive definite. However the other coefficient matrices are not necessarily positive definite. Nevertheless, if a linear position and velocity feedback loop of the form

$$u_k = u_k - (Ax_k + Bx'_k)$$

is constructed locally, equation (3.7) is reduced to the following:

$$(J_0 + H(t))x''_k + ((B(t) + B)x'_k + (A(t) + A)x_k = o(x_k) + ku_k$$

where $B = kB$ and $A = kA$. In this case, both the coefficient matrices of the velocity term and position term can be made positive definite by choosing A and B appropriately. The C^0 -Type Betterment Process converges provided every diagonal element of A and B is sufficiently large, $u_0(t)$ and $y_d(t)$ are sufficiently smooth, and their higher order terms $o(x_k)$ are ignored.

3.6 Conclusion

In this chapter we have reviewed various polynomial trajectory algorithms for robot motion planning. In particular, we have explained in detail the algorithm due to Chand and Doty[23] which uses on-line splines for joint interpolated trajectory planning. A modified version of the algorithm has been implemented, in which noise has been added to each joint of the manipulator. Experimental results which demonstrate the properties of the method have been included. We have also presented, in some detail, a second method due to Miller III [61], which uses a sensor-based control mechanism that is related to various adaptive learning schemes. Finally the "Betterment Process", proposed by Arimoto *et al* [10] has been studied and a brief overview of this scheme has been given.

We shall proceed to study the use of learning automata for trajectory planning.

Chapter 4

Methods Based on Learning Automata

4.1 Learning Automata Models

The learning automata models discussed in the literature involve interactions between automata and random environments. The interaction between the automaton and the environment can be described as follows (see Figure 4.1):

The automaton is offered a set of actions $\{\alpha_1, \dots, \alpha_R\}$ by the environment it interacts with. It is constrained to choose only one of these actions at a particular time. Once the automaton decides on an action, say α_i , this action will serve as an input to the environment. The environment will then respond to the input by giving out either a reward (the value '0') or a penalty (the value '1'), the latter being the case with the penalty probability c_i associated with α_i . This response in turn serves as an input to the automaton. Upon receiving the response from the environment, the automaton updates its internal states and/or action probability and decides on the action for the next time instant, thus repeating the process. In general, a learning automaton is one which learns the optimal action (i.e. the action which has the minimum penalty probability), and eventually chooses this action more frequently than any other action.

4.1.1 Random Environments and Learning Objectives

Suppose α_i is the action chosen by the learning automaton at the n^{th} time instant and which serves as an input to the environment. Upon receiving the input, the environment sends out a response $\beta(n)$, which as stated above

Figure 4.1 The Automaton-Environment Interaction.

has a value which is either '0' or '1'. The environment penalizes the automaton (i.e., $\beta(n) = 1$) with probability c_i , where

$$c_i(n) = \Pr[\beta(n) = 1 \mid \alpha(n) = \alpha_i] \quad \text{for } i = 1 \text{ to } R$$

Thus the characteristics of the environment are specified by the set of penalty probabilities $\{c_1(n), \dots, c_R(n)\}$. If the c_i 's are time invariant, then the environment is said to be stationary; otherwise, it is said to be non-stationary. To make the problem non-trivial, it is assumed that the set of c_i 's are unknown to the learning automaton initially.

Under the assumption that the set of penalty probabilities $\{c_1, \dots, c_R\}$ is unknown *a priori*, it is desired that as a result of interaction with the environment, the automaton arrives at the action which possesses the minimum penalty response. It may be noted that if c_L is the minimum penalty probability, and if we define

$$p_i(n) = \Pr[\alpha(n) = \alpha_i]$$

then we aim to obtain the probabilities as specified below:

$$p_L(n) = 1, \quad p_i(n) = 0 \quad \text{for } i \neq L.$$

4.1.2 Learning Criteria

With no apriori information, the automaton chooses the actions with equal probability. The expected penalty is thus initially M_0 , the arithmetic mean of the penalty probabilities.

An automaton is said to learn **expediently** if, as time tends towards infinity, the expected penalty is less than M_0 . We denote the expected penalty at time 'n' as $E[M(n)]$. The automaton is said to be optimal if $E[M(n)]$ converges to the minimum penalty probability in the limit as time goes toward infinity. It is ϵ -optimal if, in the limit, $E[M(n)] < c_L + \epsilon$ for any arbitrary $\epsilon > 0$. This could be achieved by a suitable choice of some parameter of the automaton (i.e.: the number of states of the automaton). In this case, the limiting value of $E[M(n)]$ can be as close to c_L as desired.

4.1.3 Classification of Learning Automaton

The internal decision process of a learning automaton is modeled by using a transition matrix which describes the changes of the internal state of the automaton on receiving the responses of the environment. Similarly, the action it chooses is a function of its internal state, the function being determined by an output matrix. Based on the structures of the transition matrix and the output matrix, the automata discussed in the literature can be classified mainly into two categories: Fixed Structure Stochastic Automata (FSSA), and Variable Structure Stochastic Automata (VSSA) described below.

4.2 Fixed Structure Stochastic Automata

Fixed Structure Stochastic Automata (FSSA) are those in which both the transition matrix and the output matrix are time invariant. Without loss of generality, the output matrix of an automaton can be deterministic [69]. Furthermore, if the transition matrix is also deterministic, the automaton is usually called a deterministic automaton.

Formally, a FSSA is defined as a quintuple $(\alpha, \Phi, \beta, F, G)$ where

- (i) $\alpha = \{\alpha_1, \dots, \alpha_R\}$ is the set of actions that it must choose from.
- (ii) $\Phi = \{\phi_1, \phi_2, \dots, \phi_k\}$ is its set of states, with $k \geq R$.
- (iii) $\beta = \{0, 1\}$ is its set of inputs. The input '1' represents a penalty.
- (iv) F is a map from $\Phi \times \beta$ to Φ . It defines the transition of the automaton in terms of its internal states, on receiving the input $\beta(n)$. F may be stochastic.
- (v) G is a map from Φ to α , and determines the action taken by the automaton if it is in a given state. With no loss of generality G is deterministic [69].

Tsetlin [94,95], who initiated the work on FSSA, was the first to design a machine which actually had expedient (and even ϵ -optimal) learning behaviour. It was initially called the Linear Tactic. This automaton and the Krinsky automata are two well-known deterministic automaton; the Krinsky automata will be explained in detail.

Although fixed structure stochastic automata are easy to implement, they have one major disadvantage. If an automaton is currently choosing an action, say α_i , then it will almost always, choose the same action in the next time

instant even if it receives a penalizing response from the environment. The only exception to this is usually when the automaton has reached the boundary state of α_i . In this case, the automaton will leave the current action and choose another one. This characteristic of the FSSA is not desirable when the number of states of an action is large, for this will make the automaton take a long time to leave any action, even if it is an undesirable one.

To highlight the features of the FSSA we shall now describe the Krinsky automaton.

4.2.1 Krinsky Automaton

A FSSA which had good learning characteristics, $Z_{kN,k}$ the Krinsky automaton, is defined possessing kN states $\{\phi_1, \dots, \phi_{kN}\}$ and k actions $\{\alpha_1, \dots, \alpha_k\}$ as follows :

$$Z_{kN,k} = (\{\phi_1, \dots, \phi_{kN}\}, \{\alpha_1, \dots, \alpha_k\}, \{0,1\}, Z(\cdot, \cdot), G(\cdot))$$

The individual components of $Z_{kN,k}$ are described below:

- (i) The G map is deterministic and is defined as

$$G(\phi_j) = \alpha_i \quad \text{if } (i-1)N+1 \leq j \leq iN.$$

Therefore, the automaton chooses α_i if it is in any state ϕ_j , where $(i-1)N+1 < j < iN$.

- (ii) The Z map is deterministic. The automaton can be defined as follows:

- (a) If $\beta = 0$ (obtains a favourable response), the automaton is required to move directly to the most internal state corresponding to the action chosen (i.e., $\phi_{(j-1)N+1}$ for action α_j , $1 \leq j \leq k$).

- (b) If $\beta = 1$ (i.e., it obtains an unfavourable response), the movement of

the automaton is exactly the same as that of the Tsetlin automaton. In this case it moves towards the boundary state of the chosen action one step at a time.

Thus,

$$Z(\phi_i, 0) = \phi_{(j-1)N+1} \quad \text{if } (j-1)N+1 \leq i \leq jN$$

$$Z(\phi_i, 1) = \phi_{i+1} \quad \text{if } (j-1)N+1 \leq i < jN$$

$$= \phi_{((j+1)N) \bmod (kN)} \quad \text{if } i = jN$$

In the previous equation, if $j = k - 1$, then $((j+1)N) \bmod (kN)$ will be zero. The transition map of the Krinsky automaton is shown in Figure 4.2 for the case when $k = 2$.

All Transitions with probability 1.0

Figure 4.2 $Z_{2N,2}$: The Two-Action $2N$ -State Krinsky Automaton.

The Krinsky automaton has the following property proved in Tsetlin [94].

Theorem 4.1

The Krinsky automaton is ϵ -optimal in all random environments.

♦ ♦ ♦

4.3 Variable Structure Stochastic Automata (VSSA)

A Variable Structure Stochastic Automaton (VSSA) is essentially [69] an updating rule by which the action probability vector at time $(n+1)$ is computed using the corresponding probability vector at the n^{th} time instant and the automaton-environment interaction at the same time instant.

The action probability vector $\mathbf{P}(n)$ of an R -action automaton is defined as an R -vector $[p_1(n), \dots, p_R(n)]^T$, where $p_i(n)$ is the probability of choosing action α_i at time n . Note that the sum of all the p_i 's must be unity.

A VSSA chooses an action based on the distribution specified by its action probability vector. It is implemented by using a Random Number Generator (RNG). Due to the fact that almost all the VSSA discussed in the literature permit probabilities that can take any real value between zero and one inclusive, by using an appropriate RNG the automaton can easily be made to decide on the action to be chosen. VSSA also have the added flexibility to choose two different actions at two consecutive time instants. This is because we use a RNG and the action probability vector to decide on the action to be chosen. This feature clearly overcomes the drawback of the FSSA and therefore VSSA are, in general, faster than FSSA.

Although the VSSA can be described in terms of transition and output matrices, in practice they are merely defined as action probability updating schemes which are either of a continuous [47, 67, 68, 69] or discrete nature [74, 78] because of the following theorem proven by Varshavskii and Vorontsova [67, 99].

Theorem 4.2

For every stochastic automaton in which the transition and the output matrices are updated with time, there exists an equivalent updating scheme which involves only updating of the action probability vector.

◆◆◆

Action probability updating schemes studied in the literature are mainly divided into two mutually exclusive classes: those which have **absorbing barriers**, and those which are **ergodic**. In the former class, the values of the limiting action probabilities depend on the initial distribution of the action probabilities. In the latter, however, the distribution of the limiting action probabilities is independent of the initial distribution of the values. In general, if the environment is stationary, schemes with absorbing barriers are more desirable. On the other hand, if the environment is non-stationary (i.e., the penalty probabilities are time varying), ergodic schemes are preferred because the automata will not get "locked" in any of the given actions.

Formally, a VSSA is defined as a quadruple $(\alpha, \mathbf{P}, \beta, F)$, where

- (i) $\alpha = \{\alpha_1, \dots, \alpha_R\}$ is the set of R actions offered by the environment that the automaton must choose from.
- (ii) $\mathbf{P} = [p_1(n), \dots, p_R(n)]^T$ is the action probability vector with p_i representing the probability of choosing action α_i at the n^{th} time instant, $1 \leq i \leq R$. Also,

$$\sum_{1 \leq i \leq R} p_i(n) = 1.$$

- (iii) $\beta = \{0, 1\}$ is the set of binary inputs from the environment. The input '0' represents a reward and the input '1' represents a penalty.
- (iv) F is the updating scheme. It is a map from $\mathbf{P} \times \beta$ to $\mathbf{P}$ and defines the method of updating the action probabilities on receiving an input from the

environment. If $p_i(n)$ represents the probability of choosing action α_i at the n^{th} time instant, then F is usually of the form:

$$p_i(n+1) = p_i(n) - g_i(P) \quad \text{if } \alpha_j \text{ is chosen and } \beta = 0$$

$$p_i(n+1) = p_i(n) + \sum_{j \neq i} g_j(P) \quad \text{if } \alpha_j \text{ is chosen and } \beta = 0$$

$$p_i(n+1) = p_i(n) + h_i(P) \quad \text{if } \alpha_j \text{ is chosen and } \beta = 1$$

$$p_i(n+1) = p_i(n) - \sum_{j \neq i} h_j(P) \quad \text{if } \alpha_j \text{ is chosen and } \beta = 1$$

where g and h are function of P and $i \geq 1, j \leq R$ and $i \neq j$.

In general, if $P(n+1)$ is a linear function of the vector $P(n)$, then the updating scheme is said to be linear; otherwise, it is said to be non-linear.

4.3.1 The Linear Reward-Penalty Scheme

The Linear Reward-Penalty (LPP) scheme is the earliest known updating scheme [69]. It is a linear scheme because the $P(n+1)$ term is a linear function of the components of $P(n)$ for both the reward and penalty scenarios. This scheme can be obtained by setting

$$g_i(P) = a \cdot p_i \quad \text{and} \quad h_i(P) = [b/(R-1)] - b \cdot p_i$$

Thus, the LRP scheme is defined below for $1 \leq i, j \leq R, i \neq j$:

$$\begin{aligned}
 p_i(n+1) &= p_i(n) + \sum_{j \neq i} a * p_j(n) \\
 &= p_i + a * (1 - p_i(n)) && \text{if } \alpha_i \text{ is chosen and } \beta = 0 \\
 p_i(n+1) &= p_i(n) - a * p_i(n) \\
 &= (1-a) * p_i(n) && \text{if } \alpha_j \text{ is chosen and } \beta = 0 \\
 p_i(n+1) &= p_i(n) - \sum_{j \neq i} \{[b/(R-1)] - b * p_j(n)\} \\
 &= p_i * (1 - b) && \text{if } \alpha_i \text{ is chosen and } \beta = 1 \\
 p_i(n+1) &= p_i(n) + [b/(R-1)] - b * p_i(n) \\
 &= [b/(R-1)] + (1 - b) * p_i(n) && \text{if } \alpha_j \text{ is chosen and } \beta = 1
 \end{aligned}$$

The LRP scheme has been extensively studied in the literature. The scheme is ergodic and expedient in all stationary random environments.

For the case when $R = 2$, the LRP scheme has the following simplified updating. For $i, j = 1, 2$, and $i \neq j$.

$$\begin{aligned}
 p_i(n+1) &= p_i(n) - a * p_i && \text{if } \alpha_j \text{ is chosen and } \beta = 0 \\
 p_i(n+1) &= p_i(n) + a * p_j && \text{if } \alpha_i \text{ is chosen and } \beta = 0 \\
 p_i(n+1) &= p_i(n) + b * p_j && \text{if } \alpha_j \text{ is chosen and } \beta = 1 \\
 p_i(n+1) &= (1 - b) * p_i(n) && \text{if } \alpha_i \text{ is chosen and } \beta = 1
 \end{aligned}$$

4.3.2 The Linear Reward-Inaction Scheme

The LRI scheme is similar to LRP scheme. However the difference lies in the fact that when the automaton is penalized, the probability vector is unchanged.

The LR_I scheme is defined as:

$$\begin{aligned}
 p_i(n+1) &= p_i(n) + \sum_{j \neq i} a * p_j(n) \\
 &= p_i + a * (1 - p_i(n)) && \text{if } \alpha_i \text{ is chosen and } \beta = 0 \\
 p_i(n+1) &= p_i(n) - a * p_i(n) \\
 &= (1-a) * p_i(n) && \text{if } \alpha_j \text{ is chosen and } \beta = 0 \\
 p_i(n+1) &= p_i(n) && \text{if } \alpha_j \text{ is chosen and } \beta = 1 \\
 p_i(n+1) &= p_i(n) && \text{if } \alpha_i \text{ is chosen and } \beta = 1
 \end{aligned}$$

The scheme is absorbing and ϵ -optimal in all random environments [47].

4.4 Discretized Learning Automaton

To minimize the requirement of the resolution of the random number generator and to increase the speed of convergence of the VSSA, the concept of discretizing the probability space was introduced in the literature [74, 76, 78]. As in the continuous case, a discrete VSSA is defined using a probability updating function. However, in contrast to the functions used to define continuous VSSA, discrete VSSA utilize functions that can only assume a finite number of values. These values divide the interval $[0,1]$ into a finite number of subintervals. If the subintervals are all of equal length, the VSSA automaton is said to be linear. Using these functions, a discrete VSSA can be designed and learning can be performed by updating the action probabilities in discrete steps. We describe below the Discrete Linear Reward-Penalty Automata ($DLRP$) and the Discrete Linear Reward-Inaction ($DLRI$) automata, and since we shall use the former in trajectory planning, we shall describe it in greater detail.

4.4.1 The Two-Action DL_{RP} Automata

The Two-Action Discretized Linear Reward-Penalty (DL_{RP}) automaton has $(N+1)$ states where N is an even integer. We call the set of states $S = \{s_0, s_1, \dots, s_N\}$. Associated with the state s_i is the probability i/N , and this represents the probability of the automaton choosing action α_1 . In this state the automaton chooses action α_2 with probability $(1 - i/N)$. Since any one of the action probabilities completely defines the vector of action probabilities, we shall, with no loss of generality, consider $p_1(n)$ [76].

The basic idea in the learning process is to make **discrete** changes in the action probabilities. By defining the transition map as a function from $S \times B$ to S , the changes in the action probabilities are indeed discrete. The transition map of the DL_{RP} automaton is specified by (4.1) below for $s(n) = s_k$, $1 \leq k \leq N-1$.

$$\begin{aligned}
 s(n+1) &= s_{k+1} && \text{if } \alpha(n) = \alpha_1 \text{ and } \beta(n) = 0, \\
 &&& \text{or } \alpha(n) = \alpha_2 \text{ and } \beta(n) = 1 \\
 &= s_{k-1} && \text{if } \alpha(n) = \alpha_1 \text{ and } \beta(n) = 1, \\
 &&& \text{or } \alpha(n) = \alpha_2 \text{ and } \beta(n) = 0. \quad (4.1)
 \end{aligned}$$

Observe that (4.1) is valid only for the interior states. For the end states :

$$\begin{aligned}
 s(n+1) &= s(n) && \text{if } s(n) = s_0 \text{ or } s_N \text{ and } \beta(n) = 0 \\
 &= s_1 && \text{if } s(n) = s_0 \text{ and } \beta(n) = 1 \\
 &= s_{N-1} && \text{if } s(n) = s_N \text{ and } \beta(n) = 1.
 \end{aligned}$$

Figure 4.3 shows the transition map of the automaton schematically.

Figure 4.3 Transition map of the Two-Action DLPP Automaton.

Observe that if the machine is in state s_0 it has to choose α_2 and similarly if it is in s_N it has to choose α_1 . Thus the change in action probabilities can be written for $0 < p_1(n) < 1$ as :

$$\begin{aligned}
 p_1(n+1) &= p_1(n) + 1/N && \text{if } \alpha_1 \text{ is chosen and } \beta(n) = 0 \\
 & && \text{or } \alpha_2 \text{ is chosen and } \beta(n) = 1 \\
 &= p_1(n) - 1/N && \text{if } \alpha_1 \text{ is chosen and } \beta(n) = 1 \\
 & && \text{or } \alpha_2 \text{ is chosen and } \beta(n) = 0.
 \end{aligned}$$

(4.2)

At the end states the following equalities hold:

$$\begin{aligned}
 p_1(n+1) &= p_1(n) && \text{if } p_1(n) = 0 \text{ or } 1 \text{ and } \beta(n) = 0 \\
 &= 1/N && \text{if } p_1(n) = 0 \text{ and } \beta(n) = 1 \\
 &= 1 - 1/N && \text{if } p_1(n) = 1 \text{ and } \beta(n) = 1.
 \end{aligned}$$

The way by which the action probabilities are updated gives the name of the automaton.

If $c_1 < c_2$, the automaton has no absorbing barriers except in the degenerate cases when $c_1 = 0$ or $c_2 = 1$. This implies that the Markov chain is ergodic and that the limiting distribution of the probability being in any state is independent of the corresponding initial distribution [47]. More specifically, $p_1(n)$ behaves as a homogeneous Markov chain defined by a stochastic matrix M whose arbitrary element $M_{i,j}$ is defined as :

$$M_{i,j} = \Pr[s(n) = s_j \mid s(n-1) = s_i],$$

where

$$\begin{aligned} M_{i,i-1} &= g_i c_1 + g'_i (1 - c_2) && \text{for } 1 \leq i \leq N, \\ M_{i,i+1} &= g'_i c_2 + g_i (1 - c_1) && \text{for } 0 \leq i \leq N-1, \\ M_{i,i} &= 0 && \text{for } 1 \leq i \leq N-1 \end{aligned} \quad (4.3)$$

where $g_i = i/N$ and $g'_i = 1 - i/N$. All the other elements of M are zero. Furthermore, the boundary conditions for the Markov chain are specified by :

$$M_{0,0} = (1 - c_2) \quad \text{and} \quad M_{N,N} = (1 - c_1). \quad (4.4)$$

The Markov chain consists of exactly one closed communicating class. Furthermore, since it is aperiodic, the chain is ergodic and the limiting distribution is independent of the initial distribution [43]. Let $\pi(n)$ be the state probability vector, where, for all n ,

$$\pi(n) = [\pi_0(n), \pi_1(n), \dots, \pi_N(n)]^T, \quad \pi_i(n) = \Pr [s(n) = s_i], \quad (4.5)$$

and

$$\sum_{i=0}^N \pi_i(n) = 1.$$

Then the limiting value of π is given by the vector which satisfies,

$$M^T \pi = \pi \quad (4.6)$$

Using (4.6) we now state the asymptotic properties of the DLRP automaton [81a].

Theorem 4.3 .

Let $\Delta = (c_1 + c_2 - 1)$. Then π_i , the i^{th} component of the asymptotic probability vector, obeys the following difference equation for $1 \leq i \leq N$.

$$\pi_i = \frac{c_2 - \Delta(\frac{i-1}{N})}{(1-c_2) + \Delta \frac{i}{N}} \pi_{i-1} \quad i = 1, 2, \dots, N.$$

Proof :

By definition, the limiting equilibrium probability vector π satisfies

$$M^T \pi = \pi,$$

where π is defined by (4.5) above.

Consider the stationary ergodic Markov chain defined by M [81a]. Suppose that starting at time n , we consider the sequence of states $s(n)$,

$s(n-1), s(n-2), \dots$, going backward in time. It turns out that this sequence of states is itself a Markov chain with transition probabilities $Q_{i,j}$ defined as below for state indices i and j :

$$\begin{aligned}
 Q_{i,j} &= \Pr[s(m) = j \mid s(m+1) = i] \\
 &= \frac{\Pr[s(m) = j, s(m+1) = i]}{\Pr[s(m+1) = i]} \\
 &= \frac{\Pr[s(m) = j] \Pr[s(m+1) = i \mid s(m) = j]}{\Pr[s(m+1) = i]} \\
 &= \pi_j M_{j,i} / \pi_i
 \end{aligned}$$

Furthermore, consider $\Pr [s(m)=j \mid s(m+1)=i, s(m+2)=i_2, s(m+3)=i_3, \dots, s(m+k)=i_k]$, where $j, i, i_2, i_3, \dots$, and i_k are state indices. Indeed, this is equal to $Q_{i,j}$ since,

$$\begin{aligned}
 &\Pr [s(m) = j \mid s(m+1) = i, s(m+2) = i_2, \dots, s(m+k) = i_k] \\
 &= \frac{\Pr [s(m) = j, s(m+1) = i, s(m+2) = i_2, \dots, s(m+k) = i_k]}{\Pr [s(m+1) = i, s(m+2) = i_2, \dots, s(m+k) = i_k]} \\
 &= \frac{\pi_j M_{j,i} \Pr [s(m+2) = i_2, \dots, s(m+k) = i_k \mid s(m+1) = i]}{\pi_j \Pr [s(m+2) = i_2, \dots, s(m+k) = i_k \mid s(m+1) = i]} \\
 &= Q_{i,j}.
 \end{aligned}$$

If $Q_{i,j} = M_{i,j}$ for all i, j , then the Markov chain is said to be time reversible [81a], and in this case, the stationary probabilities obey :

$$\pi_i M_{i,j} = \pi_j M_{j,i} \quad \text{for all } i, j. \quad (4.7)$$

The above implies that for all states s_i and s_j , the rate at which the process goes from s_i to s_j , which is $\pi_j M_{i,j}$, is equal to the rate at which it goes from s_j to s_i , which is $\pi_i M_{j,i}$.

In the case of the DLPP automaton, since the chain can only make transitions from any state to one of its two nearest neighbors, without the need of any extensive evaluations, it is possible to argue that the Markov chain is time reversible. The argument is as follows : Between any two transitions from s_i to s_{i+1} there must be one from s_{i+1} to s_i (and vice versa) and since the only way to re-enter s_i from a higher state is via state s_{i+1} , one can argue that for the Markov chain, the number of transitions from s_i to s_{i+1} must at all times be within 1 of the number of transitions from s_{i+1} to s_i . Thus, the rate of transitions from s_i to s_{i+1} equals the rate from s_{i+1} to s_i , rendering the process time reversible.

Thus we can easily obtain the limiting probabilities by solving for (4.7) for each state index $i=0,1,\dots,N-1$. Explicitly, for all $i=0,1,\dots,N-1$,

$$\pi_i(1 - M_{i,i+1}) = \pi_{i-1}(M_{i-1,i}). \quad (4.8)$$

But, $M_{i,i+1} = (g'_i c_2 + g_i(1-c_1))$, and $M_{i-1,i} = (g'_i c_2 + g_{i-1}(1-c_1))$, where $g_i = i / N$ and $g'_i = 1-i / N$. Equation (4.8) yields the desired result after substituting the latter values and performing some simplifying algebraic manipulations.

◆◆◆

Theorem 4.4

The $DLRP$ automaton is ϵ -optimal whenever the minimum penalty probability is less than 0.5.

Proof :

With no loss of generality let α_1 be the optimal action (i.e., let $c_1 < c_2$). It remains to be proved that $E[p_1(\infty)]$ tends to unity as $N \rightarrow \infty$ if and only if $c_1 < 0.5$, where,

$$E[p_1(\infty)] = \sum_{i=0}^N \left(\frac{i}{N}\right) \pi_i$$

We consider three mutually exclusive and exhaustive cases.

Case I : $c_2 > 0.5 > c_1$.

From (13) we can see that if $c_2 > 0.5 > c_1$, then,

$$\pi_i \geq q \pi_{i-1}$$

where $q > 1$ for all i .

This easily implies that as $N \rightarrow \infty$ the major part of the probability measure on π is contained in an arbitrarily small neighbourhood of unity. Thus,

$$\lim_{N \rightarrow \infty} E[p_1(\infty)] = \lim_{N \rightarrow \infty} \sum_{i=0}^N \frac{i}{N} \pi_i \rightarrow 1.$$

Case II : $0.5 \geq c_2 > c_1$.

$$\text{Let } i_0 = N \left[\frac{1}{2N} + \frac{1 - 2c_2}{2(1 - (c_1 + c_2))} \right].$$

Note that i_0 need not be an integer. For the sake of notation, let the ratio of π_i to π_{i-1} in (13) be q_i . Then, a simple algebraic computation shows that :

$$q_i = \frac{c_2 - \Delta(\frac{i-1}{N})}{(1 - c_2) + \Delta(\frac{i}{N})} \quad \begin{matrix} < 1 & \text{for } 1 \leq i < i_0 \\ > 1 & \text{for } i_0 < i \leq N. \end{matrix}$$

It is important to observe that for large N , $i_0 < qN$, where q is strictly less than 0.5.

The ϵ -optimality of the scheme when $c_2 = 0.5$ is disposed of by remarking that q_i increases to $(1 - c_1 + \Delta/N) / c_1$ and is strictly greater than unity for all i .

Consider now the case when $c_2 < 0.5$. In this case, $q_i < 1$ and increases for $1 \leq i < i_0$, and continues to increase beyond i_0 . We compare π_i for $0 \leq i < i_0$ and $i_0' < i < 2i_0 + 2$ to π_i for i in the interval $2i_0 + 2 < i \leq N$. In the latter interval,

$$q_i > \frac{1 - c_2}{c_2} > 1.$$

Let i_1 be the first integer in the interval $(2i_0 + 2, N]$. Then the probability measures in the first two intervals sum to a quantity **less** than $2qN\pi_1$, where q is chosen strictly less than 0.5 such that $i < qN$ for sufficiently large N and q independent of N . Similarly, the probability measure in the last interval sums to a quantity **more** than S' , where,

$$S' = ([1 - (1 - c_2/c_2)(1 - 2q)^N] / [1 - (1 - c_2/c_2)]) \pi_1 \quad (4.9)$$

This shows that for $N \rightarrow \infty$ most of the probability mass sits in the last interval, and an argument as in case (I) finishes the proof. Between cases I and II we see that the scheme is ϵ -optimal whenever $c_1 < 0.5$.

Case III : $c_2 > c_1 \geq 0.5$.

Let α be defined as $\alpha = (2c_2 - 1)/(2(c_2 + c_1 - 1))$. Of course, α is strictly greater than 0.5 in the case we are considering. Let $d > 0$ be an arbitrarily small positive number. For $i = 0, \dots, N$, let i_1 be the first of the numbers i/N which belong to the interval from $\alpha - d$ to α , and let $\pi(i_1)$ be the corresponding associated probability measure. Since the probabilities are increasing in the interval from 0 to i_1 the probability of the whole interval from 0 to $\alpha - d$ is bounded **above** by $i_1 \cdot \pi(i_1)$. We shall show that the probability of the interval from $\alpha - d$ to $\alpha - d/2$ is bounded **below** by $\pi(i_1)$ times the sum of a quotient series where the quotient is bounded **below** by $d(c_1 + c_2 - 1) + 1$ independent of N (if d is sufficiently small and N is large enough). But the number of terms in that quotient series is asymptotic to $(1/2d)N$ since each of the numbers in the intersection of the progression i/N with the interval from $\alpha - d$ to $\alpha - d/2$ contributes one term. Hence, as N tends to infinity, most of the probability mass in the interval from 0 to α sits in the interval from $\alpha - d$ to α . A very similar argument gives that most of the probability mass in the interval from α to $\alpha + d$ sits in the interval from α to $\alpha + d$, where d is arbitrarily small. This concludes the proof.

Hence the automaton is not ϵ -optimal whenever the minimum penalty

probability is greater than 0.5. Interestingly enough the value of $E[p_1(\infty)] = 1$ when $c_1 = 0.5$.

Hence the theorem I

♦ ♦ ♦

The proof of the above theorem is due to Oommen and Christensen [73].

It is well known that the Tsetlin automaton [94,95], is ϵ -optimal in environments whenever the minimum penalty probability is less than 0.5. The DLPP automaton possesses a similar property. However it is **not** a generalized version of the linear tactic, but is different in both design and operation for the following reasons :

- (i) Whereas the $L_{2N,2}$ automaton is a FSSA, the DLPP scheme is a VSSA.
- (ii) In the case of the $L_{2N,2}$ automaton, the action probability vector is a **deterministic** vector. In the case of the DLPP scheme, $p(n)$ is a random vector. Thus, whenever $c_1 < 0.5$, whereas in the former case the probability $p_1(\infty) \rightarrow 1$ as $N \rightarrow \infty$, in the latter case the **expected** probability $E[p_1(\infty)] \rightarrow 1$ as $N \rightarrow \infty$. Thus all the advantages of VSSA over FSSA (such as that of possessing the capability of choosing different actions at almost all consecutive time instances) are found in the DLPP scheme. Additionally, the expected penalty tends to the value of the minimum penalty probability whenever the latter quantity is less than 0.5.

4.4.2 The Multi-Action DLPP Automaton

The R-Action DLPP automaton operates in a discretized probability space

which divides the probability space $[0,1]$ into NR intervals when $N \geq 1$ and does not divide the probability space at all if $N=0$. As before, the action-probability vector $P(n)$ is defined by a set of probabilities, $[p_1(n), p_2(n), \dots, p_R(n)]^T$, satisfying

$$\sum_{i=1,R} p_i(n) = 1 \quad (4.10)$$

For ease of notation we omit the reference to the time instant n .

Apart from equation (4.10), we constrain each p_i such that it has to be a value on the discretized space. Let $\delta = 1 / NR$, then, apart from (4.10), p_i satisfies :

$$p_i \in \{0, \delta, 2\delta, \dots, (NR-1)\delta, 1\}$$

The probability updating rule specified by the $DLRP$ automaton in the case of a reward ($\beta(n) = 0$) is as follows for $j \neq i$:

$$\begin{aligned} p_i(n+1) &= \max(p_i(n) - \delta, 0) && \text{if } \alpha(n) = \alpha_j, \beta(n) = 0 \\ &= 1 - \sum_{j \neq i} \max(p_j(n) - \delta, 0) && \text{if } \alpha(n) = \alpha_j, \beta(n) = 0 \end{aligned} \quad (4.11)$$

The philosophy behind (4.11) is quite straightforward. If α_j is chosen and a reward is obtained, each of the other positive p_j 's is decremented by δ . These decrements are then added to $p_i(n)$.

To describe the case of a penalty response, we define a function RandVect , whose input is an integer $k < R - 1$, and j the index of an action. The output is random subset $H_k(j)$ of k indices from the set $\{1, 2, \dots, R\} - \{j\}$. Using this notation, we define the updating rule as below:

$$\begin{aligned}
p_i(n+1) &= \max(p_i(n) - (R-1)\delta, 0) && \text{if } \alpha(n) = \alpha_i, \beta(n) = 1 \\
&= p_i(n) + \delta && \text{if } \alpha(n) = \alpha_j, \beta(n) = 1, p_j(n) \leq k\delta, \\
&&& \text{where } k \geq (R-1) \\
&= p_i(n) + \delta && \text{if } \alpha(n) = \alpha_j, \beta(n) = 1, p_j(n) = k\delta, \\
&&& \text{where } k < R-1 \text{ and } l \in H_k(j)
\end{aligned} \tag{4.12}$$

The philosophy motivating (4.12) is also quite straightforward. If α_i is chosen and penalized, $p_i(n)$ is decremented by $(R-1)\delta$ if $p_i(n) \geq (R-1)\delta$, and this decrement is added to other actions, distributing the probability mass of δ for each action probability. However, if the action probability of the action chosen has only a value of $k\delta$, where $k < R-1$, this probability is decremented to zero, and k out of the $(R-1)$ remaining probabilities are randomly chosen and each incremented by δ .

The R -state $DLRP$ automaton has the following property.

Theorem 4.5

The R -state $DLRP$ automaton is expedient.

Proof:

Consider the R -state $DLRP$ automaton with $N = 0$. In this case, the vector $p(n)$ is a unit vector e_i , where e_i is the vector with zeros and a unity in the i th position. By definition if $p(n) = e_i$, the action chosen must be α_i . Thus, using (4.11) and (4.12), the probability updating rule can be written in the vector form as (4.13) below for $j \neq i$.

$$\begin{aligned}
 p(n+1) &= e_i && \text{if } p(n) = e_i, \beta(n) = 0 \\
 &= e_j && \text{with prob. } (1/R-1) \text{ if } p(n) = e_i, \beta(n) = 1,
 \end{aligned} \quad (4.13)$$

Consider the Markov chain X defined by the states $\{i | 1 \leq i \leq R\}$, where $X = i$ if a_i is the action chosen by the automaton. If A is the stochastic matrix defining the chain it can be seen that by virtue of (4.13), that A matrix has the form:

$$A = \begin{bmatrix}
 1-c_1 & c_1/R-1 & c_1/R-1 & \dots & c_1/R-1 \\
 c_2/R-1 & 1-c_2 & c_2/R-1 & & c_2/R-1 \\
 & & & & \\
 c_R/R-1 & c_R/R-1 & c_R/R-1 & & 1-c_R
 \end{bmatrix} \quad (4.14)$$

Let $\pi = [\pi_1, \pi_2, \dots, \pi_R]^T$ be the limiting asymptotic probability vector of the ergodic Markov chain described by (4.14). Clearly, π is obtained by solving

$$\pi = A^T \pi$$

π is thus the eigenvector of the eigenvalue of A which is unity.

Solving $[I - A]^T \pi = 0$ yields for the first row,

$$\{ c_1 \pi_1 - (1 / (R - 1)) \sum_{i=2}^R c_i \pi_i \} = 0$$

whence $\pi_1 = J / R c_1$, where J is a constant independent of 'i' and is equal to

$$J = \sum_{j=1}^R c_j \pi_j .$$

Similarly, $\pi_i = J / R c_i$, where J is defined above.

Since, $\sum_{i=1}^R \pi_i$ is unity, J can be solved for to have the value :

$$J = \frac{R}{\sum_{j=1}^R 1/c_j}$$

whence, $\pi_i = (1 / c_i) / \sum_{j=1}^R (1/c_j)$.

Since π_i is the limiting probability of being in state i , which is indeed the limiting probability of the automaton choosing α_i , the limiting expected penalty is $M(\infty)$, where,

$$M(\infty) = \sum_{j=1}^R \pi_j c_j = \frac{R}{\sum_{j=1}^R 1/c_j} .$$

Indeed, $M(\infty)$ is the harmonic mean of $\{c_i\}$. The expedience is proved from the above by observing that the harmonic mean of a sequence of numbers is never greater than that arithmetic mean.

♦ ♦ ♦

It can be seen that the above special case of the R-Action $DLRP$ automaton obtained by setting $N = 0$, is itself a very general (stochastic) case of Tsetlin's LR_R automaton [94, 95]. Of course, when $N \geq 1$, the resemblance between the two families of automata disappears because Tsetlin's automaton is a FSSA, and the $DLRP$ automaton is, by definition, a VSSA.

4.4.2.1 The Three-Action $DLRP$ Automata

Throughout the rest of the thesis we will merely consider the case when $R = 3$, primarily because when we consider trajectory planning, the actions imposed on a joint controller are those of going forward one step in the discretized joint space, going backwards one step and staying at the same joint angle. In the case when $R = 3$, the probability vector can be drawn graphically as a point in the plane inside an equilateral triangle. The height of the triangle is unity, and the vertices represent the unit vectors $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$. A point inside the triangle represents the vector $[p_1, p_2, p_3]^T$, where the quantity p_i is the length of the perpendicular originating from the point to the i^{th} side of the triangle.

When $N = 1$, i.e., when the probability space is divided into three equal intervals, it will yield 10 permissible probability vectors. These vectors are represented by the set of tuples (i,j,k) given below:

$$\{(i,j,k) \mid i+j+k = 3 ; i,j,k \geq 0\}$$

Note that the action probability vector represented by the tuple (i,j,k) is the vector $[i/3, j/3, k/3]^T$.

The physical representation of these tuples on the equilateral triangle described above and the transition matrix describing the Markov chain defined by the Three-Action DL_{RP} automaton are given below in Figure 4.4 and Table 4.1 respectively.

Figure 4.4 The Equilateral Triangle Representation of the Three-Action DL_{RP} Automata for $N = 1$.

	(300)	(030)	(003)	(210)	(120)	(201)	(102)	(021)	(012)	(111)
(300)	d1	0	0	0	0	0	0	0	0	c1
(030)	0	d2	0	0	0	0	0	0	0	c2
(003)	0	0	d3	0	0	0	0	0	0	c3
(210)	$c2/6 + 2d1/3$	0	0	0	$d2/3$	$c2/6$	0	$2c1/3$	0	0
(120)	0	$c1/6 + 2d2/3$	0	$d1/3$	0	$2c2/3$	0	$c1/6$	0	0
(201)	$c3/6 + 2d1/3$	0	0	$c3/6$	0	0	$d3/3$	0	$2c1/3$	0
(102)	0	0	$c1/6 + 2d3/3$	$2c3/3$	0	$d1/3$	0	0	$c1/6$	0
(021)	0	$c3/6 + 2d2/3$	0	0	$c3/6$	0	$2c2/3$	0	$d3/3$	0
(012)	0	0	$c2/6 + 2d3/3$	0	$2c3/3$	0	$c2/6$	$d2/3$	0	0
(111)	$d1/3$	$d2/3$	$d3/3$	$c3/6$	$c3/6$	$c2/6$	$c2/6$	$c1/6$	$c1/6$	0

Table 4.1 The Transition Matrix of the Three-Action DL_{RP} Automata for $N=1$.

The asymptotic efficiency of the automaton can be studied by computing the stationary probabilities of the above matrix and computing the net average penalty. This has been done for various penalty probabilities $[c_1, c_2, c_3]$. In all cases the performance of the machine is superior to the performance of the corresponding machine obtained for $N = 0$.

We conjecture that the R-action DL_{RP} is ϵ -optimal whenever $c_{\min} < 0.5$. The proof of the above conjecture will be quite involved. We believe that it will require the solution of a stochastic tensor equation. However, simulation results seem to indicate that the conjecture is true.

4.4.3 The Discretized Linear Reward-Inaction (DL_{RI}) Automaton

As before, the Two-Action Discretized Linear Reward-Inaction (DL_{RI}) automation has $(N+1)$ states where N is an even integer. Similarly to the definition of DL_{RP} scheme, the set of states are referred as $S = \{s_0, s_1, \dots, s_N\}$, and associated with the state s_i is the probability i/N . This represents the

probability of the automaton choosing action α_1 . Note that in this state the Two-Action automaton chooses the action α_2 with probability $(1-l/N)$. Since any one of the action probabilities completely defines the vector of action probabilities, we shall, with no loss of generality, consider $p_1(n)$.

The automaton does not update the action probabilities when there is a penalty response from the environment and hence is called a reward-inaction automaton [69]. It is called linear because the discretized probability values vary as a linear function of the indices of the states. For the DLR_I automaton the transition map is defined by (4.15) below for $s(n) = s_i \neq s_0$ or s_N .

$$\begin{aligned}
 s(n+1) &= s_{i+1} && \text{if } \alpha(n) = \alpha_1 \text{ and } \beta(n) = 0, \\
 &= s_{i-1} && \text{if } \alpha(n) = \alpha_2 \text{ and } \beta(n) = 0, \\
 &= s_i && \text{if } \alpha(n) = \alpha_1 \text{ or } \alpha_2 \text{ and } \beta(n) = 1.
 \end{aligned}
 \tag{4.15}$$

s_0 and s_N are absorbing states and hence if $s(n) = s_0$ then $s(n+1) = s_0$ and if $s(n) = s_N$, then, $s(n+1) = s_N$, for all n . The state transitions are shown in Figure 4.5.

Figure 4.5 The Two-Action DLR_I Automaton.

Observe that, by virtue of the probabilities associated with the states, the updating can alternatively be described in terms of the action probabilities as follows: If $p_1(n) \neq 0$ or 1 ,

$$\begin{aligned} p_1(n+1) &= p_1(n) + 1/N && \text{if } \alpha(n) = \alpha_1, \beta(n) = 0, \\ &= p_1(n) - 1/N && \text{if } \alpha(n) = \alpha_2, \beta(n) = 0, \\ &= p_1(n) && \text{if } \alpha(n) = \alpha_1 \text{ or } \alpha_2, \beta(n) = 1. \end{aligned}$$

If $p_1(n) = 0$, $p_1(n+1) = 0$ and if $p_1(n) = 1$, $p_1(n+1) = 1$.

4.4.3.1 Markovian Representation of the DLRL Automaton

It is evident from the previous description that $\{p_1(n)\}$ behaves as a homogeneous Markov chain with two absorbing states. Furthermore it is a random walk with the transition probabilities dependent on the state occupied. If $d_1 = 1 - c_1$ and $d_2 = 1 - c_2$, the transition probabilities are as follows:

$$\text{Let } P_{i,j} = \Pr[\alpha(n+1) = \alpha_j | \alpha(n) = \alpha_i]$$

Then

$$P_{i,i+1} = g_i d_1,$$

$$P_{i,i-1} = g'_i d_2,$$

$$P_{i,i} = 1 - g_i d_1 - g'_i d_2,$$

where $i \neq 0, N$ and $g_i = i/N$, $g'_i = 1 - i/N$.

$P_{0,0} = 1$ and $P_{N,N} = 1$, and all other transition probabilities are zero.

As s_0 and s_N are the absorbing states and there are no periodic states, it is clear that $p_1(n)$ tends to 0 or 1 with probability 1. However, only one of these terminal values is desired. As before, if we assume $c_1 < c_2$, then it is desired

that $p_1(n)$ should tend to 1 as in $s(n)$ to s_N . For any particular value of N , the probability of the automaton converging to s_0 or s_N can be computed by evaluating the first passage probabilities [76]. The following is the asymptotic property of the DL_{RI} scheme.

Theorem 4.5

The DL_{RI} scheme is ϵ -optimal in all random environments.

The theorem is proved by Oommen and Hansen [74].

◆◆◆

The use of the above described automata to achieve the manipulator control will be discussed in the next part of the chapter.

4.5 Manipulator Control Using Automata

The strategy by which we control the manipulator can now be proposed, since the theoretical framework has been laid.

Let us suppose we have a manipulator with K joints. These joints may be revolute or prismatic. The joint angles can be measured to yield the current position of the end effector P_i , and we assume that this is done accurately. The robot is continuously fed with a noisy version of the Cartesian coordinates of the desired goal position of the end effector $Q_f(n)$. Unfortunately, $Q_f(n)$ is noisy - and it represents the observable form of the **actual** goal position P_f , the latter itself being unknown. Our intention is to control the manipulator so that it reaches arbitrarily close to P_f . Furthermore, we intend to achieve the task adaptively.

The most straightforward method to achieve this "non-adaptively" is to take a large number of observations $Q_f(n)$ of P_f and by computing the estimate

of P_f from $\{Q_f(n)\}$, any straightforward path planning strategy (for example, one using coordinated joint interpolated moves) can be utilized to move the end effector from P_i to this estimate of P_f . We do not recommend this strategy for two reasons: First of all, this scheme is non-adaptive; secondly, in a real environment, the process of obtaining a large number of observations of P_f can be very time consuming for it could involve processing many digital images of this workspace. Indeed, the robot will have to wait while all of these images are processed - before it even starts its motion. Also, once it does start its physical motion, the time involved is again a large portion of the time that the user has to spend, because, as is well known, the mechanical motions are often the most time consuming. The ideal scenario would be if the motion was planned piecewise, and as the planned motion is executed mechanically, the computer plans the next segment of the motion. Indeed, this can be achieved, for example, with VAL II by using motion commands which are suffixed by the symbol "I".

The strategy which we propose overcomes both the above drawbacks. Let us assume that a Two-Action learning machine is used in the learning process. Every joint of the robot is equipped with such a machine, each joint independently chooses to either go forward or backward in the discretized joint space. The way by which this motion is implemented, is of course, robot dependent and is easily achieved if the motors are stepped motors.

The learning process of the manipulator is described as follows. Let $\psi(n)$ be a criterion function - for example the Euclidian distance between $Q_f(n)$ and $P_i(n)$. Based on the actions stochastically chosen by the individual automata, the position of the end effector of the robot moves to $P_i(n+1)$, see Figure 4.6. The observation $Q_f(n+1)$ is now obtained, and the criterion function computed.

If the latter function is less than it was in $\psi(n)$, each automaton is rewarded. Otherwise, the automata are penalized. Based on these responses the action probabilities are locally updated and the process is repeated. When the criterion function is small enough, the process is terminated, and a fine motion planning strategy is invoked.

In the case when the Three-Action automaton is the learning machine, each joint is equipped with such a machine, and each joint is commanded stochastically to go forward, stay where it is, or go backward at every time instant based on the action chosen by the automaton. Based on this, the position $P_i(n+1)$ of the end effector is known, and using the updated observation $Q_f(n+1)$ the criterion function is recomputed so as to subsequently reward or penalize all the automata.

Figure 4.6 Manipulator Motion Planning for Horn's 2-DOF Robot.

Our technique overcomes the drawbacks discussed above in addition, to possessing the following major advantage. The strategy **does not** require the computation of any **Inverse** kinematics - and is thus computationally extremely effective. Indeed, inverse kinematic and inverse dynamic solutions can often

have scores of parameters which are position, velocity and acceleration dependent. By permitting the automata to perform stochastically, and by repeatedly computing the straightforward criterion function, we have been able to avoid such tedious computations. Besides this, note that the automata make their decisions and update their probability vectors independently, and hence they can be made to operate in parallel - thus reducing the actual physical time that elapses.

In our implementations we have used both FSSA and VSSA automata. However, it must be noted that unlike most VSSA, maintaining the DL_{RP} and DL_{RI} automata involves incrementing (or decrementing) only one integer memory location per action, and furthermore, the process of choosing an action involves invoking a random number generator exactly once per joint.

In both the scenarios which we cited above (using Two-Action and Three-Action automata), all the automata were either simultaneously penalized or simultaneously rewarded. If each automaton should get a distinct response, one has to consider how each automaton is performing, as opposed to seeing how the collective performance of the automata is. Thus in this case, a criterion function $\psi_i(n)$ can be computed for the i^{th} joint, and the performance of this joint (in joint space) will be evaluated. But this requires - at the very least - a linearized model of the inverse kinematics and can be more expensive than the scenarios discussed above. We shall discuss the actual experimental significances of these issues in the next section.

4.6 Experimental Results

The technique that has been suggested in this thesis has been rigorously tested for a few simple two-dimensional robots. The first robot R_1 , is the familiar Horn's Robot [18] in which the robot operates in one plane, and the two joints

are revolute joints. The second robot R_2 is the 3-link generalization of Horn's Robot in which the position and the orientation of the end effector can be controlled.

Various experiments were conducted in which P_f was specified and noisy versions of P_f were generated using noise that had a Gaussian (normal) distribution. In the figures that are included (Figures 4.7 to 4.16) circles which represent perturbations of up to three standard deviations of the noise are presented to show the degree of noise introduced.

Five learning automata were used independently for the experiments namely, Krinsky, LRI, LRP, DLRI, and DLRP. Automata with two actions and three actions were independently used to control the manipulator. In each case, one hundred experiments were performed and the expected ensemble path of the robot end effector was traced. Also, the expected decrease in the Euclidian distance from $P_i(n)$ to final ideal goal position was obtained. Numerous experiments were performed using various start and goal points and using various parameters for the respective automata. Clearly, all these results cannot be presented, and so in the interest of brevity some typical results will be discussed in the rest of this section.

Table 4.2 shows the number of iterations required to converge to a point within the three standard deviation circle of the goal position. For the results given in the table, the DLRI, DLRP, and Krinsky automata had a value of NR which was 12. The corresponding parameters for LRI, and LRP automata are $a=0.9$ and $b=0.5$, and all the automata receive a single global response.

The scheme that converged fastest to the final point and stayed within the noisy area, was the one using the DLRP automaton. Experiments using distinct responses to the automata were also performed; in this case exact inverse kinematics and approximate inverse kinematics were used. Table 4.3

2

of/de

2

Micro

shows the results for DL_{RP} automata for $R=3$, and Table 4.4 for $R=2$.

Automaton	#-iterations 3 actions
Krinsky	75
L_{RP}	105
L_{RI}	195
DL_{RI}	105
DL_{RP}	75

Table 4.2 Results obtained for the Horn's 2-DOF Robot, in which each automaton had three actions and all the automata received a single global response.

Number of states	number of iterations		
	one β	two β inv. Kin.	two β App. Inv. Kin.
3	135	285	555
6	90	225	345
9	75	150	255
12	75	135	240

Table 4.3 Results obtained for the Horn's 2-DOF Robot, in which the DL_{RP} automata had three actions and the automata received the responses as given in the column headings.

Number of states	number of iterations	
	one β	two β inv. Kin.
2	90	165
4	75	150
6	60	105
8	60	75

Table 4.4 Results obtained for the Horn's 2-DOF Robot, in which the DLPP automata had two actions and the automata received the responses as given in the column headings.

For each of the automata studied we have given below typical graphs of the expected path and the decrease in Euclidian distance as a function of time. The graphs are listed as below:

- (i) Figure 4.7(a) and 4.7(b):

Experiment using Krinsky automata, for the case when $RN = 8$ and $R = 2$.

- (ii) Figure 4.8(a) and 4.8(b):

Experiment using L_{PI} automata, for the case when $a = 0.9$ and $R = 2$.

- (iii) Figure 4.9(a) and 4.9(b):

Experiment using L_{PP} automata, for the case when $a = 0.9$, $b = 0.5$ and $R = 2$.

- (iv) Figure 4.10(a) and 4.10(b):

Experiment using DL_{PI} automata, for the case when $NR = 12$ and $R = 2$.

In all these figures the standard deviation is 0.1 times the link length. The disk shows the three standard deviation range of noisy points.

The results obtained using Krinsky, LR_1 , and DLR_1 automata are not good, because if the robot uses one of these schemes, the expected trajectory does not stay within the noisy area representing the goal configuration. This is because the schemes primarily ignore the penalty responses that are obtained.

The graphs for typical scenario for the DL_{PP} are shown in Figures 4.11(a) and (b) respectively for the case when $NR = 6$ and $R = 2$. The number of iterations required to converge to a point within the three standard deviation circle of the goal position in this case is approximately 60. The decrease of the number of iterations with the increasing number of states is shown in Figure 12, table 4.3 and table 4.4. Notice that in this case, path planning is achieved in 60 iterations for the case when $NR = 8$. Similar graphs are available for the case when $R=3$. In Figure 4.13(a) we have shown the expected trajectory taken when $NR = 9$, and the decrease in the Euclidean distance is shown in Figure 4.13(b). Again, the rate of convergence of the scheme as a function of NR , with respect to the number of states, is shown in Figure 4.14. The power of the scheme is obvious - since the expected trajectory enters the three standard deviation disk in only 75 iterations for the case when $NR = 12$.

In the case when each automaton had three actions ($R=3$) and the automata received distinct responses, the responses to the automata were obtained by computing the linearized versions of the inverse kinematics. These were obtained at any instant by solving linear equations involving the "recent history" (i.e., last few Cartesian and joint space coordinates) of the end effector. The performance of the automaton is shown in Figure 4.15(a) for $NR = 9$. The corresponding decrease in the Euclidean distance from P_f as a function of n is shown in Figure 4.15(b). Surprisingly, the different responses do not improve the collective learning capability of the automata, but actually degrade it. (cf.

Figure 13(b) and Figure 15(b)).

Finally, to study the effect of the scheme for a more general robot, we have investigated its performance for the case of the three-link generalization of Horn's Robot. A variety of results have been obtained. Typically, the decrease of the number of iterations with the increasing number of states is shown in table 4.5. The convergence is shown for the case when $NR=9$ and $R=3$ in Figure 4.16(a), and the corresponding decrease in the Euclidian distance is shown in Figure 4.16 (b).

A question may be asked regarding the rationale for using the DLPP automaton in this application. It has been well-known that the updating function of a learning automaton must be dependent on the response it receives from the environment. For example, consider a continuous VSSA which **completely ignores** the penalty responses of the environment. Such an automaton is of the Reward-Inaction type, and it is well known that there are linear and nonlinear Reward-Inaction schemes which are both absolutely

Three Actions		Two Actions	
Number of States	#-iterations	Number of States	#-iterations
3	735	2	375
6	420	4	270
9	390	6	225
12	285	8	165

Table 4.5 Results obtained for the Horn's 3-DOF Robot, in which the DLPP automata have Two and Three-Action as given in the column headings.

expedient and ϵ -optimal. Apart from the continuous schemes, as shown in the Section IV [78], even discretized ϵ -optimal schemes of the Reward-Inaction flavour do exist. However, although linear symmetric Reward-Penalty schemes are at their best expedient (and definitely **not** absolutely expedient [47]), Reward-Penalty schemes are not entirely rejectable. It has been shown that by discretizing the probability space, the resulting symmetric automaton is indeed ϵ -optimal wherever $c_{\min} < 0.5$, [74]. Furthermore, apart from being ϵ -optimal, the automaton utilizes **all** the information provided by the environment, and thus does not ignore any response as a Reward-Inaction automaton does.

The power of the scheme suggested in this chapter is that the sensing mechanism used need not be too sophisticated. A primitive sensor with such a learning strategy could provide excellent results.

Figure 4.7a Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Two-Action Krinsky automata with $RN = 8$ and with a single response controlling both the automata.

Figure 4.7b Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.7a.

Figure 4.8a Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Two-Action L_{RI} automata with $\alpha = 0.9$ and with a single response controlling both the automata.

Figure 4.8b Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.8a.

Figure 4.9a Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Two-Action L_{RP} automata with $a = 0.9$, $b = 0.5$ and with a single response controlling both the automata.

Figure 4.9b Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.9a.

Figure 4.10a Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Two-Action DL_{PI} automata with $NR = 12$ and with a single response controlling both the automata.

Figure 4.10b

Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.10a.

Figure 4.11a

Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Two-Action DLPP automata with $NR = 6$ and with a single response controlling both the automata.

Figure 4.11b Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.11a.

Figure 4.12 Expected number of iterations for the robots to enter the three standard deviation disk of the final point for cases when the automata use a single response.

Figure 4.13a

Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Two-Action $DLRP$ automata with $NR = 9$ and with a single response controlling both the automata.

Figure 4.13b Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.13a.

Figure 4.14 Expected number of iterations for the robots to enter the three standard deviation disk of the final point for cases when the automata use a single response.

Figure 4.15a Expected Path (given as a sequence of points) of the two-link Horn's Robot with links of length unity. The control is achieved using the Three-Action DLR_P automata with $NR = 9$ and with separate response for each automata, where these responses are obtained using a first order approximation of inverse Kinematics.

Figure 4.15b

Expected change in distance from the initial configuration to the final mean configuration for the set-up described in Figure 4.15a.

Figure 4.16a Expected Path (given as a sequence of points) of the Three-link Horn's Robot with links of length unity. The control is achieved using the Three-Action DL_{PP} automata with $NR=9$ and with single response controlling both the automata.

Figure 4.16b

Expected change in distance from the initial configuration to the final mean configuration, for the set-up described in Figure 4.16a.

4.7 Conclusions

In this chapter we have introduced the elementary concepts of the theory of learning automata. The automaton aims to learn the optimal action offered by the environment with which it interacts. We have briefly discussed the two families of learning automata studied in the literature, namely the Fixed Structure Stochastic Automata (FSSA) and the Variable Structure Stochastic

Automata(VSSA). Five illustrative automata have been described: the Krinsky, the LRP, the LRI, the DLRP and the DLRI.

We have also considered the problem of controlling the manipulator arm in which P_i , the initial position of the end effector, is known and the goal position P_f is noisily sensed. The robot is required to move from P_i to P_f , but instead of having P_f specified, a series of noisy observations $\{Q_f(n)\}$ of P_f are available. The solution which we have presented involves using learning automata. The various automata mentioned above are located at the joints and they operate in parallel to control the individual joints of the robot.

All the five automata discussed in this chapter have been used in the path planning phase of the manipulator. The Krinsky, LRI and DLRI automata perform very poorly. Once they come within the neighbourhood of the expected goal point, they overshoot the point and are unable to recover to their goal. This is because these automata tend to ignore penalty responses (or weight penalty responses with little credibility, as in the case of the Krinsky automaton). Automata which processed reward and penalty responses would definitely be the ones to be used in this setting. In lieu of this, the LRP and the DLRP automata are the best possible candidates. However, since the symmetric LRP is at its best only expedient, and the DLRP has the ϵ -optimal property (in restricted environments), the latter will be the most superior automaton of the five which have been studied for the present problem. This theoretical argument is indeed verified by the experimental results that have been presented in this chapter.

Chapter 5

Autonomous Mobile Robots

5.1 Chapter Statement

Wide utilization of computer control technology and the automation of control systems have in recent years prompted the development of mobile robots. These robots are intended for load transportation and collection of information about the terrain with the minimal participation of the human operator in the control process. Several types of guidance systems have been required to achieve this goal.

An essential task for a mobile robot system is that of autonomous navigation and motion control of mobile robot. This will be the theme of this chapter. In the first section a review of previous solutions available for navigation and the motion control of mobile robots is given. Following this, three algorithms are explained in detail. The first algorithm due to Olivier and Ozguner [72], uses a laser rangefinder sensor. The second, due to Mitchell [62], considers an autonomous vehicle navigating in a 2-dimensional world. This algorithm has been simulated in the course of the thesis and results of the simulation of this algorithm are shown. Finally an algorithm by Iyengar *et. al* [44] in which the vehicle uses a learned spatial graph, and an incremental Voronoi diagram is obtained.

5.2 Literature Review and Previous Solutions

Research on mobile robots began in the late sixties and early seventies, possibly with the Stanford Research Institute's pioneering work during which time two versions of Shakey, an autonomous mobile robot, were built in 1968 and 1971. The main purpose of this project was "to study processes for the

realtime control of a robot system that interacts with the complex environment" [71].

One of the oldest and most common themes in motion planning work is the idea of representing the problem in such a way that the object to be moved is a point. Moravec[64], for example, describes path planning for the Stanford Cart. The Cart and the obstacles in its environment are approximated by enclosing them in circles. The circles bounding these objects are then increased in radius by the radius of the circle enclosing the robot, and the robot's position is represented by the center of its enclosing circle. A path for this point that avoids the enlarged obstacles is sought.

This theme is closely related to that of the point representation of a moving object in a space which is not necessarily the two or three-dimensional space of the physical problem. This space is often called the configuration space. Udupa[97, 98] solves obstacle avoidance problems for a manipulator by using a configuration space in which the moving object is represented by a moving point.

Once configuration space obstacles (or their approximations) have been obtained, there are several strategies for finding a path for the moving reference point that represents the object, say, A. One method in two-dimensions (without rotations) is to construct a visibility graph whose vertices are the Cartesian space representations for the initial and final configuration of A together with points on the Cartesian space obstacle boundaries. The edges of this graph represent the existence of straight line paths (or obstacle edges) of legal positions of the moving object in the Cartesian space. Path planning is thereby converted to searching for a path between two vertices of the visibility graph.

Another approach is to find a decomposition into cells of the set of points in Configuration space that represent the legal, collision-free configurations of the moving object. Then a graph is constructed whose vertices and edges represent these cells and their adjacencies. This approach has yielded a good

exact algorithm (Lozano-Perez[50]) that finds shortest paths for the special case of a polygon with fixed orientation being translated in the plane. An approximate algorithm was also presented (Brooks[19]) for the case when rotations and translations are permitted. A practical difficulty with the former approach is that it finds paths that "hug" obstacle boundaries.

A third approach is to have the point representing the object move along edges (or curves) of the generalized Voronoi diagram. The paths in this case are safe and usually lie in the interior of the free-space as opposed to being on the boundaries of the configuration space obstacles.

Schwartz and Sharir [86, 87] showed the possibility of using analytical and topological rather than purely geometric methods in motion planning.

In practice, manipulator uncertainty is a serious problem. Also, errors may arise from unforeseen events as well as from the positioning limitations of the robot and from the inaccuracies in the environment model. Consequently, sensing the environment is essential, even when the environment is in principle well-known. Vision strategies rely on large amounts of data that can support some kind of statistical pattern recognition approach. Gaston and Lozano-Perez [31] and Grimson and Lozano-Perez [34] study the problem of identifying and locating objects using sparse range or tactile data that provide approximate surface points and their normals. These papers focus on the problem of generating feasible interpretations of the data and testing these interpretations for consistency with the object models. Schwartz and Sharir [87] and Bernstein [15] suggest tactile probing algorithms that involve choosing a next probe direction based on the contact locations of the previous probe.

There has not been too much work reported in the literature with respect to robot navigation in an unexplored terrain. This problem can be attributed, at least in part, to the lack of global information about the obstacles and their location. Many [46] of the existing solutions to this problem are based on sensors information [22, 33, 64, 90], and in general do not achieve global

optimality for navigation paths. We shall now describe an algorithm which uses a Laser Rangefinder to perform this navigation.

5.3 Algorithm due to Oliva and Ozguner

The choice of sensors for a particular vehicle has a profound impact upon the vehicle's ability to navigate autonomously. The laser rangefinder sensor has been developed by the Environmental Research Institute of Michigan (ERIM), for use on the Adaptive Suspension Vehicle. The latter was developed at The Ohio State University. This sensor, if used subsequent to some preprocessing functions, is able to yield an elevation map of the area in front of the vehicle in a short period of time.

The navigation algorithm due to Olivier and Ozguner [72], which is the major component of this section, can be subdivided into four parts: region classification, path planning, path smoothing and path execution.

5.3.1 Region Classification

This section identifies all dangerous obstacles in front of the vehicle. The problem of deciding which areas are unsafe for the vehicle to travel, ultimately rests with the vehicle's operating characteristics. Regions are classified as belonging to the following set; peak, pit, ridge, ravine, flat area, safe hill, unsafe hill, and this classification is based upon topographical structures. The determination of a region class is based upon the gradient and the second directional derivatives around a particular area of elevation. To obtain these, a least square fit of a cubic over two variables is calculated from the elevation data and using this cubic model the quantities can be calculated.

The values of the gradient and the second directional derivatives determine the label given to the point. Let e_1 be equal to the maximum second derivative and let l_1 be a unit vector in the direction of e_1 . Also, let e_2 be equal to the minimum second directional derivative and let l_2 be a unit vector in that

direction. With these quantities the classification of the different regions is achieved using the following thresholds:

peak	=>	$e2, e1 < 0, V_f = 0$
pit	=>	$e2, e1 > 0, V_f = 0$
ridge	=>	$abs(e2) > abs(e1), l1 * V_f = 0$
ravine	=>	$abs(e1) > abs(e2), l2 * V_f = 0$
flat area	=>	$e1 = e2 = V_f = 0$
safe hill	=>	$l1 * V_f \neq 0, l2 * V_f = 0, V_f \neq 0, V_f < \tan \alpha$
unsafe hill	=>	$l2 * V_f \neq 0, l1 * V_f \neq 0, V_f \neq 0, V_f > \tan \alpha$

In the above, α is the safest angle the vehicle can climb.

Typical results produced with the range data is shown in Table 5.1.

10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	-10	10	10	10	10	10	40	10	10	10	10	10	10	10
10	10	10	-10	10	10	10	10	10	10	40	10	10	10	10	10	10
10	10	10	-10	10	10	10	10	10	10	10	40	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	40	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	40	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	30	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	30	10	10	10	10	10	10	10	10	10	-20	10
10	10	10	10	30	10	10	10	10	10	10	10	10	10	10	-20	10
10	10	10	30	10	10	10	10	10	10	10	10	10	10	10	-20	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	-20	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10

Table 5.1 Typical output of the classification obtained using the Laser Rangefinder.

5.3.2 Path Planning

In this section we will discuss how to construct a simple path around the obstacles. The classification map generated in the previous section, along with

the vehicle's characteristics, defines an obstacle map -which is a map of the dangerous areas that the vehicle must avoid.

To find the simple path for the vehicle, this program generates points termed as subgoals that define a series of line segments from the goal to the vehicle. The algorithm basically searches for gaps between the obstacles.

The algorithm is divided in various modules, as follows:

- (i) The gap generation module finds gaps by making an imaginary scan around the present position of the search, calculating the distance to obstacles in the area. If, at any time, the difference between two successively calculated distances is greater than a threshold value, this position defines the start of a gap.
- (ii) The gap elimination module works with the imaginary scan data. After a gap has been identified, its width is determined. If it is not large enough to allow the vehicle to pass through easily, this gap is eliminated. A simple cost function, called the deviation function [60], is applied to the remaining gaps which is the angle formed by the line joining the present goal position to the vehicle and the line joining the goal position and the gaps.

After the best gap has been chosen, the search position is moved into this gap, and the process is repeated, unless the vehicle is visible. If the vehicle is visible, then the search is complete, and the generated subgoals define a set of line segments from the goal to the vehicle.

5.3.3 Path Smoothing

The path smoothing section of the algorithm accomplishes two functions. First, it must smooth the path generated by the path planning so that it is kinematically possible for the vehicle to execute the path, and second, it must generate the turning and velocity commands needed by the pilot software.

The algorithm works as follows. First, the goal for the obstacle avoidance

section is chosen. Next, a simulated scan is performed at discrete angles around the vehicle, generating data needed by the kinematics routine. It is this procedure that guides the vehicle by calculating a new velocity vector for the vehicle to follow for the next sampling period of the simulation.

Finally, the algorithm forms a set of relative measures of safety for the vehicle to travel in certain directions. The vector that has the largest component in the direction of the goal is chosen as the new direction for the vehicle to travel in this sampling period. Since these vectors are based upon kinematic quantities, the new velocity of the vehicle can be calculated from the magnitude of these vectors.

The calculation of the safety coefficients begins with the calculation of a set of velocities, V_{oi} , which are the velocities at which the vehicle approaches obstacles surrounding it. The magnitude of these velocities is the component of the vehicle's present velocity in the direction of the obstacle.

The next step is to calculate the stopping distance for the vehicle if it were travelling at the above mentioned velocities. If B is equal to the maximum braking acceleration of the vehicle, then the distance in which the vehicle can stop, D_{stop} , is found by the following equation:

$$D_{stop} = \|V_{oi}\|^2 / 2 * \|B\|$$

With this distance in mind, the set of safety coefficients, $\{S_i\}$, can be defined, where, S_i is defined in the direction A_i , in terms the distance D_{stop} , and the magnitude of A_i as below:

$$S_i = A_i - A_i * D_{stop} / \|A_i\|$$

Another quantity calculated is called the Desirability, D_i , which is the direction of the chosen subgoal. Let G be a unit vector in the direction of the chosen subgoal. Then D_i can be defined as follows:

$$D_i = S_i * G$$

The new direction, A_i , is the direction that gives the maximum value of D_i , where, the new velocity can be calculated. It is a function of the sampling interval of the simulation and the damping factor, K , and obeys:

$$\|V_{\text{new}}\| = \max[D_i] / [T \cdot K].$$

With this new velocity the vehicle can be moved until it has reached the goal location and the routine is terminated.

The navigation system was tested by Olivier and Ozguner [72]. It was shown to produce fast and reliable results in areas containing a high percentage of obstacles.

5.4 Mitchell's Algorithm

In this section an algorithm due to Mitchell [62] will be described. Also the results of its implementation will be presented. This implementation was done as part of the work done in this thesis.

Consider an autonomous vehicle navigating in a 2-dimensional world of arbitrary obstacles. Assume that the robot can be modelled as a circular, omnidirectional vehicle of radius R . The vehicle has its current location $P = (x, y)$. Assuming that the vehicle is equipped with a sensor, it will provide a range scan of distances, $r(\theta)$, to the closest obstacles at angle θ in the field of view.

The sensor has a finite range w , meaning that if the distance to the closest obstacle at angle θ is greater than or equal to w , it will return $r(\theta) = w$. Otherwise, it simply returns the distance $r(\theta) < w$. The vehicle stops to take a new scan every time it reaches a new decision point or when it has travelled D units since its last scan ($D \leq w$).

The map is the knowledge the vehicle has about its surroundings. When the system starts its navigation, the map may be empty or may contain some a

priori knowledge of the vehicle environment. As the vehicle moves, it scans the terrain and acquires knowledge about the obstacles that it may encounter. The vehicle also has the ability to forget information by deleting from the map data which has become inaccurate or irrelevant.

The data representations used by Mitchell [62] involve the concepts of walls, corridors of free space, and points, called candidate subgoals. Walls are polygonal representations of the barriers presented by obstacles. Corridors of free space are passageways between walls which appear to be unobstructed, and subgoals are locations within a corridor to which the vehicle may move in order to pass through the corridor. The path traversed refers to the list of points along the path that the vehicle has executed since it left the start position. The map at any instant in time is defined by the triple (W, C, G) , where W is the list of walls generated by the current scan, C is the list of corridors derived from W , and G is the corridor connectivity graph.

The node set for G consists of all path points P_i in the path encountered. It also consists of all subgoals that were generated at each such point. Arcs in G connect each P_i with each subgoal generated at P_i , and connect a subgoal S_i (generated at P_i , associated with corridor C_i) with a path point P_j , only if P_j is visible through corridor C_i from P_i . Basically, G is a graphical representation of what the vehicle has seen so far. Its nodes indicate locations where the vehicle has been (P_i) or it has considered going to (S_i), and its arcs correspond to connections between regions of free space. The role of G is to keep track of the options that have been available to the robot and to approximate the topology of the free space that it has encountered. Information can be discarded from the map by deleting from G those nodes (and the arcs that are incident on them) that lie more than a given distance δ from the current vehicle location P .

Given a range scan $r(\theta)$, the vehicle first processes it into a geometrical

interpretation of the surroundings. This involves inferring the existence of obstructing walls and of passable gaps.

The general rule for determining walls involves looping through the scan sequentially by increasing the angle θ . Consider a pair of consecutive points $r_1(\theta_1)$ and $r_2(\theta_2)$. If $r_1(\theta_1) < w$, $r_2(\theta_2) < w$, and $|r_1 - r_2| < 2R$, then the points r_1 and r_2 are points on obstacles and the vehicle cannot pass between them. Thus it asserts that a wall segment exists between these points. Otherwise, there may be a passable gap between r_1 and r_2 . It must then look ahead in the scanned area to see if there exists a scan point $r_3(\theta_3)$, with $r_3(\theta_3) < w$ and $|r_1 - r_3| < 2R$. If such a point exists, it connects r_1 and r_3 with a wall segment and continues processing from r_3 ; otherwise, r_1 marks the endpoint of a wall (which we record in a list), and it continues processing from r_2 . When it reaches the end of the scan, it has completed the task. The result is a list of piecewise-linear walls, W , each wall being a list of scanned points.

For each corridor of free space, the vehicle determines points called candidate subgoals, which represent positions to which the center of the vehicle may move in order to travel through a given corridor. Each candidate subgoal S is labelled with a number $h(S)$ which represents a guess based on the map of the distance from S to the goal. It then selects the subgoal S which minimizes $|P - S| + h(S)$. The vehicle then moves towards the selected subgoal, stopping only upon reaching its destination point.

5.4.1 Implementation

The above algorithm was implemented by us during the course of this study. In this study, the obstacles were represented as polygons. Instead of sensors yielding the distance to the object, the intersection of a scan line with

the polygons was implemented and the Euclidean distance was calculated. The angular resolution used was 3 degrees.

In order to find a subgoal, the definition of $h(S)$ used was the straight line distance from S to the goal, ignoring any obstacles that have already been seen and that may block the path from S to the goal. This is certainly a lower bound on the path length from S to the goal.

Numerous experiments were performed for a variety of obstacles and paths. Since all these results cannot be presented here, for the sake of brevity we shall give an example in which the terrain has two obstacles (see Figure 5.1). The position of the robot is shown by the larger circle, and the goal by the smaller circle. Figure 5.2 shows the path to the subgoal, and finally Figure 5.3 shows the path taken by the robot from the start to the goal positions. Such results are typical.

Figure 5.1 An implementation of Mitchell's [62] algorithm. The robot is represented by the larger circle, and the goal is represented by the smaller circle.

Figure 5.2 Path taken by the robot from start to a subgoal position for the set-up described in Figure 5.1.

Figure 5.3 Path taken by the robot from start to the goal position for the set-up described in Figure 5.1.

5.5 A Robot Navigation Algorithms Using Learned Spatial Graphs

In this section we will describe in detail the algorithm due to Iyengar *et. al* [46], in which the robot is treated as a point and only navigation in two dimensions is considered. Learning begins by characterizing the terrain information, which is modelled using an attribute graph called a spatial graph, and a Voronoi diagram, which is gradually built by consolidating the information about the path traversed up till that time. Initially, the graph is empty, and the obstacle avoidance techniques use local optimization for the navigation of the robot. By local optimization, the authors of [46] mean that "optimal" paths are selected based **only** on sensor information at the time of the decision. Learning is incorporated by augmenting the terrain model with the information about new paths of navigation. The paths are locally optimal initially, and become globally optimal as learning proceeds.

5.5.1 Local Optimization of Object Avoidance

The local optimization of the obstacle avoidance is achieved by considering the obstacle that is nearest to the source point S. The sensor readings are obtained, which include the distance from the source to the edges of the obstacle, and also the corresponding edge angles relative to the line between the robot center and the goal. In the local optimization stage two cases can be considered.

Figure 5.4 First Case for Local Optimization of Iyengar's *et al* Algorithm.

The first case is the one in which, the obstacle does not extend beyond the source point in the direction opposite to the direction of the destination points (see Figure 5.4). The local optimization criterion is to minimize the distance traversed in the direction perpendicular to the line joining the source point S, to the destination point D. That is, using the notation of Figure 5.4, the locally optimal path is given by the condition $\min(d_1, d_2)$ or $\min(p_1 \sin \alpha_1, p_2 \sin \alpha_2)$. Observe that this method may not yield a globally optimal path.

The second case of local optimality involves obstacles that extend beyond the source in the direction opposite to the direction of the destination point as shown in Figure 5.5. In this case the distance traversed in the direction opposite to that of the destination point D also has to be minimized. The criterion for local optimization is the minimum of the pair $(\sqrt{d_1^2 + f_1^2}, \sqrt{d_2^2 + f_2^2})$. The following procedure describes the local optimization.

Figure 5.5 Second Case for Local Optimization of Iyengar's *et. al* Algorithm.

procedure Navigate-Local (S,D);

Input: The source point S and the destination point D

Output: go to commands

begin

1. **If** (D is directly reachable) **then**

2. go straight;

3. **else**

4. Scan the terrain around the directo of SD;

5. $P^* \leftarrow$ optimum (P1,P2);

6. go straight to P^*

7. **If** (P^* is not equal, D) **then**

8. Navigate-Local (P^* ,D);

endif;

endif;

end;

5.5.2 Terrain Model

The terrain in which the robot navigates is represented by both a spatial graph and a Voronoi diagram. The spatial graph G is defined as the ordered triple (V, E, Ψ) , where V is the set of nodes, $V \times V \supseteq E$ is the set of edges, and $\Psi: V \rightarrow \mathbb{R}^2$ is an attribute mapping that defines a pair of attributes for each vertex.

For an edge $e = (v_i, v_j) \in E$ we say that v_i and v_j are connected to each other.

Each edge $e = (v_i, v_j) \in E$, also has a distance $d(e) = [(i_1 - j_1)^2 + (i_2 - j_2)^2]^{1/2}$, where $\Psi(v_j) = (j_1, j_2)$.

Initially a uniform grid is superimposed on the terrain of navigation. The granularity or grid size is chosen to be smaller than the size of the smallest obstacle. The grid cells are numbered in the usual manner using an x and y coordinate system. Any path of navigation on the grid consists of straight lines and stop points. Each stop point corresponds to a node of the spatial graph, and each path joining two adjacent stop points corresponds to an edge. The pair of attributes of a node corresponds to the coordinates of the grid cell in which the node lies. The distance of an edge, $e = (v_i, v_j) \in E$, is the Euclidean distance between the nodes v_i and v_j .

A Voronoi diagram is obtained for the set of vertices V of the spatial graph G . By definition, the Voronoi diagram of a set of points $S = \{p_1, p_2, \dots, p_n\}$, $\text{Vor}(S)$, partitions the plane into n equivalence classes, each of which corresponds to a point. Specifically, the equivalence class corresponding to the point p_i is the Voronoi polygon $VP(p_i)$, defined as,

$$VP(p_i) = \{r / r \text{ in } \mathbb{R}^2, \text{ and } d(r, p_i) \leq d(r, p_j) \text{ for } j > i\}.$$

Initially, when the robot is first placed in a new terrain, the spatial graph is empty (or null) and the Voronoi diagram contains no points. The new paths are integrated into the terrain models as and when they are traversed. The spatial

graph is updated for every new path as follows:

- (i) create new nodes corresponding to the new stop points encountered,
- (ii) create new edges corresponding to the paths in between two adjacent stop points,
- (iii) create new intersection nodes, corresponding to the intersection points of new edges with the existing edges,
- (iv) break the unexplored regions into obstacles, free, and unexplored regions, i.e., combine the adjacent free regions into a single free region.

Initially the obstacles are bounded by polygons much bigger than the obstacles. But as learning proceeds the bounding polygons shrink and eventually converge to the shape of the obstacles. The Voronoi diagram is again updated to include the newly inserted nodes.

5.5.3 Path Planning and Learning

In this section, an algorithm that plans safe paths to navigate from a new arbitrary source point to a new arbitrary destination point is developed.

Consider the navigation of the robot from the source point S to the destination point D . A virtual source S' and a virtual destination point D' are computed, such that $S \in VP(S')$, and $D \in VP(D')$. In other words S' , and D' are the nodes of the spatial graph that are nearest to S and D respectively. The paths from S to S' , and D' to D are traversed according to the local optimization criterion. The learning corresponding to these paths, involves the creation of new nodes for the stop points and the appropriate edges. Also the Voronoi diagram should be updated by creating new Voronoi regions for the new nodes.

The process of finding the virtual points should be carried out only after the graph is reasonably complete. Thus, until a considerable number of nodes are inserted into the spatial graph the navigation will, at its best, be locally

optimal. The basic procedure is as follows:

procedure Navigate (S,D);

Input: The source point S and the destination point D

begin

1. Find S' and D' such that $S \in VP(S')$, and $D \in VP(D')$;
2. Navigate-Local (S, S');
3. Navigate-Global (S' , D');
4. Navigate-Local (D' ,D);

end;

The procedure Navigate (S' , D'), plans the path from S' to D' . This procedure tests the polygon P containing the source end of S' to D' . A polygon is said to be an obstacle-polygon with respect to S' , if the obstacle or obstacles contained in P entirely fill the sensor range from S' . A polygon is a free polygon if it does not contain any obstacles.

procedure Navigate-Global (S', D');

Input: The source point S' and the destination point D' , on the spatial graph
 $S'D'$ stands for the straight line joinings S' and D' .

begin

```

1. Find the polygon  $P$ , that contains source end of  $S' \rightarrow D'$ ;
2. If ( $P$  is an OBSTACLE-POLYGON) then
3.     Find the nearest intersection point  $S$ , of  $S'D'$  and  $P$ ;
4.     Find  $S^*$ , such that  $S \in VP(S^*)$ ;
5.     MOVE TO  $S^*$  along edges of  $P$ ;
6.     Navigate-Global ( $S^*, D'$ );
7. else
8.     If ( $P$  is a FREE-POLYGON) then
9.         Find the intersection point  $S$ , of  $S' \rightarrow D'$  and  $P$ ;
10.        GO directly to  $S$ ;
11.        Navigate-Global ( $S, D'$ );
12.    else
13.        If ( $P$  is unexplored with respect to  $S'$ ) then
14.            Explore ( $P, S'$ );
15.            Consolidate ( $P, S'$ );
16.            Navigate ( $S', D'$ );
17.        endif;
18.    endif;
19. endif;
20. end;

```

If the polygon P is unexplored with respect to S' , the procedure Explore(P, S') is invoked. The polygon contains two types of regions, visible and invisible. The invisible regions are the regions of the polygon that are not reachable by the sensor. These regions are declared as unexplored with respect to the vertices on the line that limits the range of sensor from S' . The visible regions are then partitioned into obstacle polygons and free polygons.

procedure Explore (P,S);

Input: P is the polygon to be explored with respect to vertex S.

begin

1. Partition P into visible and invisible regions;
 2. **For** each invisible region **Do**
 3. declare the region to be unexplored with respect to the new vertices;
 4. Partition the visible region into obstacle regions and free-regions;
 5. **For** each free-region R **Do**
 6. declare R to be free with respect to the vertices of R;
 7. **For** each obstacle-region R **Do**
 8. declare R to be obstacle-region with respect to S;
- end;**

Finally, the procedure Consolidate given below checks for any adjacent free regions forming a convex region. If they, in fact, form a convex region, the procedure combines them to create a single free polygon.

procedure Consolidate (P,S);

Input: P is an explored polygon with respect to vertex S.

begin

1. **For** each free-polygon P_1 belonging to the partition P **Do**
 2. find all adjacent free-polygons of P_1 ;
 3. Find the maximal subset of them that forms a convex polygon and combine them into a single polygon;
- end;**

It can be concluded from the overall Navigation procedure that the learning is incorporated along with path planning. The paths are locally optimal at first, and as learning proceeds, they gradually become globally optimal.

This procedure was one of the first to introduce the concept of learning in the domain of robot navigation and movement. A subsequent procedure will be discussed in the next chapter.

5.6 Conclusions

In this chapter we have studied the problem of autonomous mobile robot navigation. Initially, a brief literature survey was given and subsequently three algorithms were reviewed in detail. The first algorithm due to Olivier and Ozguner [72], uses a laser rangefinder sensor to perform the navigation using a topological representation of the terrain. The second, due to Mitchell [62], considers an autonomous vehicle navigating in a 2-dimensional world, in which the vehicle scans the terrain and acquires knowledge about the obstacles. The data representation involves the concepts of walls, corridors of free space, and candidate subgoals and is achieved using two lists and a connectivity graph respectively. This algorithm has been implemented in the course of this study and the results have been included. Finally, an algorithm due to Iyengar *et al.* [44] which uses a Learned Spatial graph, and a Voronoi diagram has been described in fair detail.

Chapter 6

Path Planning with Learning

In this chapter we consider the process of navigation in a completely unexplored obstacles terrain, which involves both learning the information about the obstacles terrain and path planning. The algorithm, due to Oommen *et. al* [75], solves the problem of navigating the robot and simultaneously learning the terrain model. The navigation process is constituted by a number of traversals, where each traversal is from an arbitrary source point to an arbitrary destination point. At every stage of learning, the partially learnt terrain model is represented as a Learned Visibility Graph, and it is updated after each traversal.

6.1 Notations and Definitions

The method described in this chapter assumes that the robot begins the navigation in a completely unexplored terrain of finite dimensions. The robot is treated as a point. The terrain is populated with stationary disjoint and convex polygonal obstacles. Let $W = \{w_1, w_2, \dots, w_k\}$ be the set of obstacles. The aim is to have terrain computed and maintained as a Learned Visibility Graph (LVG).

To obtain the LVG, the robot initially navigates through the obstacles using a Local Navigation technique. This technique, which is a "hill climbing technique", is shown to converge by Oommen *et. al* [75]. In the process of local navigation, the robot updates the LVG. It is shown that the LVG ultimately converges to the actual Visibility Graph (VG) of the obstacle terrain. Throughout the learning stage, the robot relies heavily on its sensing mechanism to perform the navigation. Subsequently, once the VG has been built the robot can

navigate without sensing, by merely planning its path using the information maintained in the VG.

Formally, the VG is a pair (V, E) , where

- (i) V is the set of vertices of the obstacles,
- (ii) E is the set of edges of the graph. A line joining the vertices v_i to v_j forms an edge $(v_i, v_j) \in E$ if and only if it is an edge of an obstacle or it is not intercepted by any other obstacle. Formally $(v_i, v_j) \in E$ iff (i) $(v_i, v_j) \in Z$ or (ii) $(v_i, v_j) \cap Z = \emptyset$.

The intermediate stages of learning are captured in terms of the Learned Visibility Graph (LVG) which is defined as follows: $LVG = (V^*, E^*)$, where $V \supseteq V^*$ and $E \supseteq E^*$. The LVG is initially empty and is incrementally built.

The robot is equipped with sensors capable of measuring the distance to an obstacle in any specific direction. Also, the robot is equipped with sensors which enables the navigation along the edges of the obstacle.

In terms of notation, the straight line from the point P to the point Q is denoted by $P \rightarrow Q$. η_{PQ} denotes the unit vector along the straight line $P \rightarrow Q$.

6.2 Local Navigation and Learning

The Local Navigation technique enables the robot to detect and avoid obstacles along the path from an arbitrary source point, S , to an arbitrary destination point, D .

The path of the robot is completely decided by the sensors' reading. The obstacles in the proximity of the source point are scanned and a suitable path of navigation is chosen: the information acquired during the Local Navigation is integrated into the partially built terrain model.

We assume that the robot is equipped with a primitive motion command

Move (S, A, λ) where, S is the source point where the robot is located, A is the destination point which may or may not be specified, λ is the direction of motion, which is always specified.

If A is specified, then the robot moves from S to A in a straight line path. In this case, the direction of motion λ is the vector η_{SA} , the unit vector in the direction of $S \rightarrow A$. If A is not specified, then the robot moves along the direction λ as follows: If the motion is alongside an edge of an obstacle, then the robot moves to the end point of the edge along the direction λ . This end point is returned to the calling procedure as point A . If the motion is not alongside an edge of an obstacle, then the robot traverses along the direction λ till it reaches a point on the edge of an obstacle. This point is returned as the point A to the calling procedure.

The outline of this procedure is as follows. The robot moves along $S \rightarrow D$ until it gets to the nearest obstacle. It then circumnavigates this obstacle using a Local Navigation strategy. The technique is then recursively applied to reach D from the intermediate point. Further, apart from path planning the procedure also incorporates the learning phase of acquiring the VG.

In the Local Navigation technique, the robot moves along the direction η_{SD} until it encounters an obstacle at a point A which is on the obstacle edge joining the two vertices, say, A_1 and A_2 . At this point the robot has two possible directions of motion: along AA_1 or AA_2 . A local optimization criterion J , is defined as follows:

$$J = \eta_{SD} \cdot \lambda \quad (6.1)$$

where λ is the direction of motion.

Let λ_1 and λ_2 be the unit vectors along AA_1 and AA_2 respectively. Let $\lambda^* \in \{\lambda_1, \lambda_2\}$ maximize the function J given in equation (6.1) .

The robot then undertakes an exploratory traversal along the direction $(-\lambda^*)$ until it reaches the corresponding vertex called the exploratory vertex. At this exploratory point the terrain is explored using the procedure Update-Vgraph. Then the robot retraces along the locally optimal direction λ^* until it reaches the other vertex S^* , whence it again calls Update-Vgraph procedure. The procedure Navigate-Local is recursively applied to navigate from S^* to D .

The procedure Update-Vgraph implements the learning component of the robot navigation. Whenever the robot reaches a new vertex v_i this vertex is added to the LVG. From this vertex, the robot beams its sensor in the direction of all the existing vertices of the LVG. The edge (v_i, v_j) is added to the edge set E^* , corresponding to each vertex $v \in V^*$ visible from v_i .

procedure Update-Vgraph (v); —

Input: The vertex v which is newly encountered.

output: The updated LVG = (V^*, E^*) . Initially the LVG is set to (ϕ, ϕ) .

comment: $DIST(v_1, v_2)$ indicates the Euclidean distance between vertices v_1 and v_2 , if they are visible to each other. This is the auxiliary information stored along with the LVG.

begin

1. $V^* = V^* \cup \{v\};$
2. **for** all $v_1 \in V^* - \{v\}$ **do**
3. **If** (v_1 is visible form v) **then**
4. $DIST(v_1, v) = |v_1 \rightarrow v|;$
5. $E^* = E^* \cup \{(v_1, v)\};$
6. **else**
7. $DIST(v_1, v) = \infty;$
- endif;**

end;

The procedure Navigate-local uses the primitive motion command MOVE and the procedure Update-Vgraph during execution. This procedure is formally described as follows:

```

procedure Navigate-Local (S,D);
Input:    The source point S and the destination point D
Output:  A sequence of elementary MOVE commands
begin
1.   if (D is visible from S) then
2.       MOVE(S,D,  $\eta_{SD}$ )
3.   else
4.       if (S is on an obstacle and the obstacle obstructs its view) then
5.           compute{  $\lambda_1, \lambda_2$  }, the two possible directions of motion;
6.            $\lambda^*$  = direction maximizing  $\lambda_i \cdot \eta_{SD}$ ;
7.           if (S is a vertex) then
8.               if ( $S \notin V^*$ ) then Update-Vgraph (S);
9.               MOVE (S,S*,  $\lambda^*$ );
10.          else
11.              MOVE(S,S1,  $\lambda^*$ ); {make exploratory trip to S1}
12.              if ( $S_1 \notin V^*$ ) then Update-Vgraph (S1);
13.              MOVE(S1,S*,  $\lambda^*$ ); {retrace steps to S*}
14.              if ( $S^* \notin V^*$ ) then Update-Vgraph (S*);
15.              endif;
16.          endif;
17.          Navigate-Local (S*,D);
18.          else {move to next obstacle}
19.              MOVE(S,S*,  $\eta_{SD}$ ); {move to next obstacle along  $\eta_{SD}$ }
20.              Navigate-Local (S*,D);
21.          endif;
22.      endif;
23.  endif;
24. end;

```

The proof that the procedure Navigate-Local converges is shown in detail in the original paper [75].

6.3 The Backtracking Facility

When the obstacles touch the terrain boundaries the procedure Navigate-Local is not guaranteed to halt. There are two possible cases in this scenario:

- (i) There is no path between the source point S and the destination point D . In this case, a single obstacle blocks all the paths from S to D .
- (ii) The angle between the obstacle and the terrain boundary is less than $\pi/2$, in which case there is a path between S and D . In such a case the robot may be forced to move to the dead corner formed by the obstacle and terrain boundary resulting in a cul-de-sac.

To take care of such dead corners, backtracking is imparted to the procedure Navigate-Local. The robot backtracks whenever it reaches a point from which no further moves are possible. The robot backtracks along the edges of the obstructing obstacle until an edge (S, S_1) that makes an angle less than $\pi/2$ with η_{SD} is encountered. The fact that such an edge (S, S_1) exists is guaranteed because of the convexity of the obstacles. If for the same obstacle, the robot has to backtrack twice, then there is no path between S and D .

The procedure Backtrack is given below:

procedure Backtrack (S,D,S*);

Input: The point D is the destination point.

S is a dead corner, i.e., a vertex of an obstacle and is also on the boundary of the terrain.

Output: A sequence of MOVES from S in such a way that if a path exists, then it can be determined using Navigate-Local. The location S* is returned to the calling procedure.

begin

1. $S_1 = S$;

2. λ^* = only permitted direction of motion on the obstacle;

3. **while**(S $\rightarrow$ D. $\lambda^* < 0$) **do**

4. MOVE(S_1, S^*, λ^*);

5. $S_1 = S^*$;

6. λ^* = only permitted direction of motion on the obstacle;

end;

6.4 Global Navigation

In the local navigation stage the paths taken are not necessarily globally optimal from the path planning point of view. However the extra work carried out in the form of learning is inevitable because of the lack of information about the obstacles. The LVG is gradually built as a result of learning, and in the regions where the visibility graph is available, the optimal path can be found by computing the shortest path from the source point to the destination point on the graph without the use of sensors. The computation can be carried out in quadratic time in the number of nodes of the graph by using Dijkstra's algorithm[2].

The global navigation maximizes the use of the available LVG in planning navigation paths. In the regions where no LVG is available, the procedure Navigate-Local is used for navigation. In these regions the LVG is updated for future navigation. The outline of the global navigation is as follows:

```
procedure Navigate-Global (S,D);  
begin  
1. Compute-Best-Vertices ( $S^*, D^*$ );  
2. Navigate-Local-With-Backtrack (S, $S^*$ );  
3. Move-On-LVG( $S^*, D^*$ );  
4. Navigate-Local-With-Backtrack ( $D^*, D$ );  
end;
```

The closeness of the paths planned by Navigate-Global to the globally optimal path depends on the degree to which the LVG is built. The paths tend to be globally optimal as the LVG converges to the VG. The proof that the LVG indeed converges to VG after a sufficiently large number of calls of Navigate-Local, is shown in detail in the original paper [75].

6.5 Implementation and Results

The algorithm described above was implemented, along with the various modifications which permitted the robot to backtrack. Various experiments using various obstacle terrains were conducted.

A typical example of the implementation is given below. Figure 6.1a shows a terrain with four obstacles; the position of the robot is represented by a big circle and the goal position is represented by a small circle. Figure 6.1b shows the path (dotted line) taken by the robot, as specified by Navigate-Local. Figure 6.1c shows the corresponding LVG obtained after one traversal. Figure 6.2a shows a second initial and goal positions, Figure 6.2b the corresponding path and Figure 6.2c the corresponding LVG. Similar scenarios are shown in Figure 6.3a, Figure 6.3b and Figure 6.3c.

Figure 6.1a The obstacle terrain for the experiments. The robot is represented by the larger circle, the goal position is represented by the smaller circle.

Figure 6.1b First Path taken by the robot in the set-up of Figure 6.1a.

Figure 6.1c The LVG after executing the Navigate-Local and the motion described in Figure 6.1b.

Figure 6.2a The robot is represented by the larger circle, and second the goal position is represented by the smaller circle.

Figure 6.2b Second Path taken by the robot in the set-up of Figure 6.2a.

Figure 6.2c The LVG after executing the Navigate-Local and the motion described in Figure 6.2b.

Figure 6.3a The robot is represented by the larger circle, the third goal position is represented by the smaller circle.

Figure 6.3b Third Path taken by the robot in the set-up of Figure 6.3a.

Figure 6.3c The LVG after executing the Navigate-Local and the motion described in Figure 6.3b.

To study the rate by which the LVG converges to the VG, experiments were conducted in which the goal position in each traversal was randomly generated. One hundred experiments were performed and the expected number of edges learned as a function of the number of traversals was recorded. The results are shown in the Table 6.1. The corresponding learning graph is given in Figure 6.4. Note that after only 20 traversals 81.9 % of the total number of vertices and 77 % of the total number of edges were learned. The power of the scheme is obvious.

# goal	vertices	edges
1	2.05	1.62
2	3.88	4.27
3	5.42	7.07
4	6.72	9.17
5	8.09	12.54
6	9.14	15.14
7	10.04	17.69
8	10.70	19.65
9	11.44	21.71
10	11.90	23.22
11	12.54	25.36
12	13.08	27.20
13	13.71	29.45
14	14.18	31.27
15	14.68	32.99
16	15.04	34.28
17	15.33	35.13
18	15.62	36.14
19	16.04	37.70
20	16.38	39.17

Table 6.1 Expected number of vertices and edges learned using random goal points for the obstacle terrain shown in Figure 6.1a.

Figure 6.4 Graphical plot of the Table 6.1.

6.6 Conclusion

In this chapter we have described the navigation algorithm for a mobile robot due to Oommen *et. al* [75]. The algorithm involves both learning the information about the obstacles terrain and path planning. The navigation process is defined by a number of traversals, where each traversal is between an arbitrary source and a destination point. At every stage of learning, the partially learnt terrain model is represented using the Learned Visibility Graph, which has been shown to converge asymptotically to the exact Visibility Graph.

A typical result of a simulation of the algorithm has been presented. The results which demonstrate the rate by which the LVG converges to the VG, are presented.

Chapter 7

Conclusions

7.1 Summary of the Results

In this thesis we have considered the robot motion problem given the initial and desired final positions of a robot. We have considered the problem from the points of view of two types of robots. Initially we studied algorithms that pertained to a manipulator robot. Subsequently, the autonomous mobile robots were studied, and in this case it was assumed that they had no knowledge of the environment but were constantly informed by a variety of sensors.

With regard to manipulators arms, we have reviewed polynomial trajectories, described and studied by Paul [81]. We presented some experimental results of an algorithm due to Chand and Doty [23]. This is an algorithm which uses cubic splines and a two-point look-ahead strategy for joint interpolated motion. We also reviewed in some detail, the Cerebellar Model Articulation Controller (CMAC), which was adapted to robotics by Miller III [62]. In this CMAC model, information learned during previous movements is used to achieve new control goals as long as it involves similar regions of the system state space. Various learning control systems were also reviewed, and in particular one due to Arimoto *et. al* [10] was studied. Their learning control method is based on a simple iterative algorithm called the "Betterment process", which updates the control input based on the data obtained from the previous motion operation and improves its performance for the next operation.

In this context we have proposed a solution of our own for the problem of controlling a manipulator arm in which P_i the initial position of the end effector is known, and the goal position is noisily sensed. The robot is required

to move from P_i to P_f , but instead of having P_f specified, a series of noisy observations, $\{Q_f(n)\}$, of P_f are available. The solution which we have proposed involves using learning automata. The automata are stationed at each joint of the robot, and these operate in parallel to control the individual joints of the robot.

We have given some experimental results involving various automata, and, from these we conclude that the algorithm that uses the DLPP automata converges faster and more accurately than those which use other automata. We have alluded to the facts that the Two-Action DLPP automaton is ϵ -optimal wherever the minimum penalty probability is less than 0.5, and that the general R-action DLPP automaton is expedient in all random environments. It is conjectured to be ϵ -optimal in environments when the minimum penalty probability is less than 0.5.

In the second part of the thesis we have considered navigation algorithms for mobile robots equipped with sensors. An algorithm using the Laser Rangefinder has been described [72]. We have also studied and implemented the algorithm due to Mitchell [62], which assumes a sensor input in the form of a range scan of distances to closest obstacles. These distances are obtained at a variety of angles in the vehicles' field of view. Finally, we have described the algorithm of Iyengar *et al.* [44], in which the robot gradually builds the terrain model using a Learned Spatial graph and a Voronoi diagram.

In greater detail the path planning algorithm using Learned Visibility Graphs has also been presented [75]. In this algorithm the robot builds the terrain model as it navigates, and stores the processed sensor information in terms of a Learned Visibility Graph which is a subset of the entire obstacle Visibility Graph. The terrain is completely learned when the entire Visibility graph of the obstacle terrain has been constructed. The algorithm has been

implemented and some results which demonstrate the learning capability of the scheme has been included.

7.2 Future Research

The following are the future research possibilities that we envisage:

- (i) The scheme presented in Chapter four used the DLPP automata to achieve the motion planning of the manipulator from the initial to the destination point. One open problem is to actually study the power of the scheme for a real life robot which has very primitive sensing operations.
- (ii) Preliminary simulation results using various learning automata seem to suggest that the method proposed in Chapter four could lead to fascinating linear motion (or dog-chase) strategies. It is likely that these may not involve extensive inverse kinematic solutions. This problem remains open.
- (iii) Two new families of discretized Reward-Penalty automata have been recently discovered which are ϵ -optimal in all random environments [73]. The study of the use of these automata to achieve the path planning more effectively is a new possibility.
- (iv) It is well known that both decomposition and retraction are universal approaches to motion planning, and are indeed even tractable for many basic planning problems. Unfortunately, the general techniques associated with these approaches are often inefficient. There is certainly considerable room for inventiveness in producing computationally attractive algorithms in these areas. For example, the

concept of using Voronoi diagrams as retracts, is attractive because they possess elegant theoretical properties. However, computing these diagrams is not always done in a straightforward way. We expect that the design of increasingly efficient algorithms for specific robot systems will continue to be an area of much study both in the practical and in the theoretical domain of research. Moreover, problems in three dimensions are still mostly open.

References

1. Ahlberg, J.H., Nilson, E.N., and Walsh, J.L., "The Theory of Splines and their Applications", New York Academic Press, 1967.
2. Aho, A., Hopcroft J., and Ullman J., "The Design and Analysis of Computer Algorithms", Addison-Wesley, Reading, Mass., 1974.
3. Albus, B., "Behaviour and Robotics", Peterborough, NH : Byte Books, 1981, ch 6, pp. 139-179.
4. Arimoto, S., Kawamura, S., and Miyazaki, F., "Bettering Operation of Robots by Learning", Journal of Robotic Systems 1-2, 1984, pp. 123-140.
5. Arimoto, S., Kawamura, S., and Miyazaki, F., "Can Mechanical Robots Learn by Themselves?", Proc. of the 2nd. Int. Symp. on Robotics Research, Kyoto, Japan, August 1984.
6. Arimoto, S., Kawamura, S., and Miyazaki, F., "Bettering Operation of Dynamic Systems by Learning: A New Control Theory for Servomechanisms of Mechatronics Systems", Proc. of the 23rd. IEEE CDC, Las Vegas, Nevada, December 1984.
7. Arimoto, S., Kawamura, S., and Miyazaki, F., "Iterative Learning Control for Robotic Systems", IECON'84, Japan, 1984, pp. 393-398.
8. Arimoto, S., Kawamura, S., and Miyazaki, F., "Applications of Learning Methods for Dynamic Controls of Robot Manipulators", Proc. of 24th. IEEE CDC, Fort Lauderdale, Florida, December 1985.
9. Arimoto, S., Kawamura, S., and Miyazaki, F., "Can Mechanical Robots Learn by Themselves?", Robotics Research: The Second Int. Symposium (ed. by H. Anafusa & H. Inove) MIT-Press, Cambridge Massachusetts, 1985, pp. 127-134.
10. Arimoto, S., Kawamura, S., Miyazaki, F., and Tamaki S., "Learning Control Theory for Dynamical Systems", Proc. of the 24th. CDC, 1985, pp. 1375-1379.
11. Arimoto, S., and Takegaki, M., "A New Feedback Method for Dynamic Control of Manipulators", Trans. ASME: Journal Dynamic Systems, Measurement and Control, 1981, 103(2), pp. 126-133.

12. Arimoto, S., "Mathematical Theory of Learning Controls ", Proc. of the 4th Yale Workshop on Applications of Adaptive Systems Theory, New Haven, Ct, May 1985.
13. Atkeson, C. G., and McIntyre, J., "Robot Trajectory Learning Through Practice", Proc. of the IEEE Robotics and Automation Conf., April 1986, pp. 1737-1742.
14. Azadivar, F., " The Effect of Joint Position Errors of Industrial Robots on their Performance in Manufacturing Operations", The Journal of Robotics and Automation, Vol. RA. 3 No. 2, April 1987.
15. Bernstein, H.J., "Determining the Shape of a Convex n-side Polygon by using $2n+k$ Tactile Probes", New York University Courant Institute of Math Science, TR 125, 1984.
16. Birk, J.R., "Computation for Robotics to Orient and Position Hand Held Workpieces", Trans. IEEE Sys., Man and Cyber., SMC-6(10), 1976, pp. 665-671.
17. Bolliger, J., and Duffie, "Computer Algorithms for High Speed Continuous Path Robot Manipulator ", ANN. CIRP 28(1), pp. 391-395.
18. Brady, M., Hollerbach, J., Johnson, T., Lozano-Perez, T., and Mason M., "Robot Motion: Planning and Control", MIT-Press, 1982.
19. Brooks, R.A., and Lozano-Perez, T., "A Subdivision Algorithm in Configuration Space for Path with Rotation", IEEE Trans. on Systems, Man and Cyber., Vol. SMC-15, No.2, March/April 1985, pp. 224-233.
20. Kim, B.K., and Shin, K.G., "Minimum-time Path Planning for Robot Arms and Their Dynamics", IEEE Trans. on Systems and Cyber., Vol. SMC-15 No. 2, 1985, pp. 213-223.
21. Clenshaw, C.W., and Negus, B., "The Cubic X-Spline and its Applications to Interpolation ", Journal of Inst. of Math. Appl., Vol. 22, 1978, pp. 109-112.
22. Chattergy, R., "Some Heuristics for the Navigation of a Robot", The Int. Journal on Robotics Research, Vol. 4, No. 1, Spring 1985, pp. 59-66.

23. Chand, S., and Doty, K.L., "On-Line Polynomial Trajectory for Robot Manipulator", The Int. Journal on Robotics Research, Vol. 4, No. 2, Summer 1985.
24. Craig, J.J., and Raibert, M.H., "Hybrid Position/Force Control of Manipulators", Trans. ASME: Journal on Dynamic Systems, Measurement and Control, 1981, 103(2), pp. 126-133.
25. Craig, J.J., "Adaptive Control of Manipulators Through Repeated Trials", Proc. of the 1984 American Control Conf., 1984, pp. 1566-1576.
26. DeBoor, C., "A Practical Guide to Splines ", New York: Springer-Verlag, 1978.
27. Derby, S. J., "Simulating Motion Elements General Purpose Robot Arms", Int. Journal of Robotics Research, 1983, 2(1):3-12.
28. Dubowsky, S., Norris, M. A., and Shilleer, Z., "Time Optimal Trajectory Planning for Robotic Manipulator", Proc. of the IEEE Robotics and Automation Conf., April 1986, pp. 1906-1912.
29. Finkel, R.A., "Constructing and Debugging Manipulator Programs ", Ph.D. Dissertation, Stanford University, 1976.
30. Fu, K.S., "Learning Control Systems --Review and Outlook", IEEE Trans. on Automation Control, Vol. AC-16, 1970, pp. 210-221.
31. Gaston, P.C. and Lozano-Perez, T., "Tactile Recognition and Localization Using Object Models: The Case of Polyhedra on a plane", IEEE Tran. on Pat. Anal. and Machine Int., V. PAMI-63, May 1984, pp. 257-266.
32. Gerelle, E. G. R., "Force Feedback Control", Proc. of the 4th Int. Conf. on Industrial Robot Technology, Stuttgart, Germany, May 30-June 1, pp. 194-205.
33. Giralt, G., Sobek, R., and Chatila, R., "A Multilevel Planning and Navigation System for a Mobile Robot ", Proc. of the 6th Int. Conf. on Artificial Intelligence, August 20-23, Tokyo, 1979, pp. 335-338.
34. Grimson, W. and Lozano-Perez, T., "Model-Based Recognition and Localization from Sparse Range or Tactile Data", MIT AI Lab., AI MEMO 738, August 1983.

35. Groover, M., Nagel R., Odrey M., and Weiss Mitchell, "Industrial Robotics: Technology Programming, and Applications", Mc. Graw-Hill, 1986.
36. Hara, S., Omata, T., and Nakano, M., "Synthesis of Repetitive Control Systems and Its Application ", Proc. of the 24th. CDC, Fort Lauderdale, Florida, December 11-13, 1985.
37. Hayati, S.A., "Robot Arm Geometric Link Parameter Estimation", Proc. of the 22nd. IEEE Conf. on Decision and Control, December 1983, pp. 1477-1483.
38. Hanafusa, H., and Asada, H., "Adaptive Control of a Robot Hand Equipped with Pneumatic Proximity Sensors", Proc. of the 3rd Conf. on Industrial Robot Technology and 6th Int. Symposium on Industrial Robots, University of Nottingham, England, March 24-26, 1976, pp.D4 12p.
39. Herman, M., "Fast, Three-Dimensional, Collision-Free Motion Planning", Proc. of the 1986 IEEE Int. Conf. in Robotics, pp. 1056-1063.
40. Hewitt, J.R., "Decoupled Control of Robot Movement", Electronics Lett, 1979,15(21), pp. 670-671.
41. Hewitt, J.R., "Dynamic Decoupled Control of Robot Arms", in Proc. of the IEEE Colloquium on Control of Manipulators and Robotic Devices, London, April, pp. 4.
42. Hollerbach, J. M., "Dynamic Scaling of Manipulator Trajectories", Journal Dynamic Systems, Measurement and Control, 1984, 106:103-106.
43. Hornick, M.L., and Ravani, B., "Computer-Aided Off-Line Planning and Programming of Robot Motion", The Int. Journal of Robotics Research, Vol. 4, No. 4, Winter 1986.
44. Iyengar, S.S., Jorgensen, C. C., Rao, S.V.N., and Weisbin, C.R., "Robot Navigation Algorithms Using Learned Spatial Graphs", ORNC Technical Report Tm-9782, Oak Ridge National Laboratory, Oak Ridge, August 1985.
45. Iyengar, S.S., Jorgensen, C.C., Rao, S.V.N., and Weisbin, C.R., "Learned Navigation Paths for a Robot in Unexplored Terrain", Proc. of the 2nd Conf. on Artificial Intelligence Applications and Engineering of Knowledge Based Systems, Miami Beach, Florida, December 11-13, 1985.

46. Iyengar, S.S., Jorgensen, C.C., Rao, S.V.N., and Weisbin, C.R., "Concurrent Algorithms for Autonomous Robot Navigation in an Unexplored Terrain", Proc. of the IEEE Robotics and Automation Conf., April 1986, pp. 1137-1150.
47. Lakshmivaraham, S., "Learning Algorithms Theory and Application", Springer-Verlag, New York, 1981.
48. Lewis, R.A., "Autonomous Manipulation on a Robot Summary of Manipulator Software Functions ", TM-33-679, Pasadena Jet Propulsion Laboratory, California Institute of Technology, 1974.
49. Lipkin, H., and Duffy, J., "Analysis of Industrial Robots Via Theory of Screws", 12th Int. Symp. on Industrial Robots., 1982.
50. Lozano-Perez, T., and Wesley, M.A., "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles", Commun. ACM, Vol. 22, No. 10, October 1979, pp. 560-570.
51. Lozano-Perez, T., "Spatial-Planning: A Configuration Space Approach", IEEE Trans. on Computers, V.C-32, February 1983, pp. 108-120.
52. Loeff, L.A., and Soni, A.H., "Algorithm for Computer Guidance of a Manipulator Between Obstacles", ASME paper No.74-Det-89, 1974.
53. Luh, J.Y.S., "An Anatomy of Industrial Robots and Their Controls", Tutorial on Robotics, 1983, pp. 5-23.
54. Luh, J.Y.S., Lin, C.S., and Chang, P.S., "Formulation and Optimization of Cubic Polynomial Joint Trajectories for Industrial Robots", IEEE Trans. on Automatic Control, AC-28 (12), 1983.
55. Luh, J.Y.S., and Walker, M.W., "Minimum Time Along the Path for a Mechanical Arm", in Proceedings of the 16th IEEE Conf. on Decision and Control, New Orleans, LA, December 7-9, 1977, pp. 775-759.
56. Lumelsky, V., "On Non-Heuristic Motion Planning in Unknown Environment.", Proc. of the IFAC Symposium on Robot Control (Syroco '85) Barcelona, Spain, November 1985.
57. Lumelsky, V., "Continuous Path Planning for Various Planar Robot Arm Configurations ", Yale University Tech. Report, No. 850, April 1985.

-
58. Masterson, J.W., Poe, E.C., and Fardo, S.W., "Robotics", Department of Industrial Education and Technology Eastern Kentucky University, Preston Publishing Company Inc., 1985.
 59. McDonald, A., "Robot Technology: Theory, Design and Applications", Prentice-Hall, 1986.
 60. McKinsson, J.E., "Guidance and Navigation for Intelligent Mobile Autonomous Systems", Masters Thesis, The University of Florida, 1983.
 61. Miller III, T.W., "Sensor-Based Control of Robotic Manipulators Using a General Learning Algorithm", The Journal of Robotics and Automation, Vol. RA 3 No. 2, April 1987.
 62. Mitchell, J.S.B., "An Autonomous Vehicle Navigation Algorithm ", Applications of Artificial Intelligence, 1984, pp. 153-158.
 63. Mita, T., "Repetitive Control of Mechanical Systems", Proc. of ATACS'84, Japan, Oct. 4-5 1984, pp. 1-4.
 64. Moravec, H.P., "The CMU Rover" Proc. of the National Conf. in Artificial Intelligence, August 1982, pp. 377-380.
 65. Moravec, H.P., "The Stanford Cart and the CMU Rover ", Proc. in IEEE, V71, July 1983, pp. 872-884.
 66. Murata, H., "Kinematic Design for Articulated Robot Arm", Engineering Faculty, Robe University, Japan, No. 22, 1976, pp 83-92
 67. Narendra, K.S., and Thathachar, M.A.L., "On the Behaviour of a Learning Automaton in a Changing Environment with Routing Applications", IEEE Trans. on Syst. Man and Cyber., Vol. SMC-10, 1980, pp. 262-269.
 68. Narendra, K.S., Wright, E., and Mason, L.G., "Application of Learning Automata Telephone Traffic Routing", IEEE Trans. on Syst. Man and Cyber., Vol. SMC-7, 1977, pp. 785-792.
 69. Narendra, K.S., and Thathachar, M.A.L., "Learning Automata --A Survey ", IEEE Trans. Syst. Man and Cyber., Vol. SMC-4, 1974, pp. 323-334.
 70. Narendra, K.S., and Shapiro, I.J., "Use of Stochastic Automaton for Parameter Self-Optimization with Multimodal Performance Criteria", IEEE Trans. on Systems Science and Cyber., Vol. 5, 1969, pp. 352-360.

-
71. Nilson, N.J., "A Mobile Automaton: An Application of Artificial Intelligence Techniques", Proc. of the 1st IJCAI Washington, D.C., May 1969, pp. 509-520.
 72. Olivier, J.L., and Ozgumer, F., "A Navigation Algorithm for an Intelligent Vehicle with a Laser Rangefinder", Proc. of the IEEE Robotics and Automation Conf., April 1986, pp. 1145-1150.
 73. Oommen, B.J., and Christensen, J.P.R., "On Three Families of Asymptotically Optimal Linear Reward-Penalty Learning Automata", Proc. of the 1987 IEEE Int. Conf. on Sys., Man and Cyber., Alexandria, Virginia, October 1987, pp. 923-928.
 74. Oommen, B.J., and Hansen E., "The Asymptotic Optimality of Discretized Linear Regard-Inaction Learning Automata", SCS-TR-36, January 1984.
 75. Oommen, B.J., Iyengar, S.S., Rao S.V.N., and Kashyap, R.L., "Robot Navigation in Unknown Terrains using Learned Visibility Graphs. Part I: The Disjoint Convex Obstacle Case", SCS-TR-86, February 1986.
 76. Oommen, B.J., Iyengar, S.S., and Andrade N., "Trajectory Planning of Robot Manipulators in Noisy Workspaces using Stochastic Automata", Topic to be presented at the Fourth IEEE Conf. on A.I. Applications, 1988.
 77. Oommen, B.J., and Thathachar, M.A.L., "Discretized Reward-Inaction Learning Automata", Journal of Cyber. and Information Science, Spring 1979, pp. 24-29.
 78. Oommen, B.J., "Absorbing and Ergodic Discretized Two-Action Learning Automata", IEEE Trans. on Syst. Man and Cyber, Vol. SMC-16, 1986, pp. 282-296.
 79. Paul, R., "Modeling Trajectory Calculation and Servoing of a Computer Controller Arm", Stanford Artificial Intelligence Laboratory, Stanford University, AIM 177, 1972.
 80. Paul, R., "Robot Manipulator, Mathematic Programming and Control", The MIT-Press, Cambridge, Massachusetts and London, England, 1981.
 81. Paul, R., "Manipulator Path Control", Proc. of the 1975 Int. Conf. on Cyber. and Society IEEE, New York, September 1975.

-
- 81a. Ross, S. M., "Introduction to Probability Models", Academic Press, New York, 1980.
 82. Raibert, M.H., "Mechanical Arm Control Using a State Space Memory", SME Tech. Paper Ser. Manuscript for Robots 2 Conf., Detroit MI, October 31-November 3, 1977, Oowe Bi, NS77-750.
 83. Saridis, G.N., and Valavanis, K.P., "Mathematical Formulation of the Organization Level of an Intelligent Machine", Proc. of the IEEE Robotics and Automation Conf., April 1986, pp. 267-272.
 84. Shahinpoor, M., "A Robot Engineering Textbook", Harper & Row Publishers Inc., 1987.
 85. Stokic, D., and Vukobratovic, M., "One Engineering Concept of Control of Manipulators", Trans. ASME. Journal in Dynamic Sys., Measurement and Control, 103(2), pp. 119-125.
 86. Schwartz, J.T., and Sharir, M., "On the Piano Movers' Problem : I. The Special Case of Rigid Polygonal Body Moving amidst Polygonal Barriers", Comm. Pure Appl. Math, Vol. 36, 1983, pp. 345-398.
 87. Schwartz, J.T., and Sharir, M., "On the Piano Movers' Problem: II General Techniques for Computing Topological Properties of Real Algebraic Manifold", Adv. Appl. Math 4, 1983, pp. 298-351.
 88. Shwartz, J.T., "Some Remarks on Robot Vision", TR 119, New York Univ. Courant Institute of Math Science, April 1984.
 89. Taylor, R., "Planning and Execution of Straight Line Manipulator Trajectories", Tutorial on Robotics 1985, pp. 152-164.
 90. Thompson, A.M., "The Navigation System of the JPL Robot ", Proc. of the 5th Int. Joint Conf. in Artificial Intelligence, August 22-25 1977, Cambridge, Mass., pp. 749-757.
 91. Togai, M., and Yamano, O., "Learning Control of Robot Manipulators", SIAM Conf. on Geometric Modeling and Robotics, Albany, N.Y., July 15-17, 1985.
 92. Togai, M., and Yamano, O., "Learning Control and Its Optimality", Proc. of the IEEE Robotics and Automation Conf., April 1986.

-
93. Tsyppkin, T., "Adaptation and Learning in Automatic Systems ", New York Academic Press, 1971.
 94. Tsetlin, M.L., "On the Behaviour of Finite Automata in Random Media", Automat. Telemek, Vol. 22, 1961, pp. 1345-1354.
 95. Tsetlin, M.L., "Automaton Theory and Modelling of Biological Systems", Academic Press, New York and London, 1973.
 96. Uchiyama, M., "Formation of High Speed Motion Pattern of Mechanical Arm by Trial", Trans. on the Society of Instrument and Control Engineer, Vol. 19-5, 1978, pp. 706-712 .
 97. Udupa, S., "Collision Detection and Avoidance in Computer Controlled Manipulators", in Proc. 5th. Int. Joint Conf. Artificial Intelligence, Massachusetts Inst. Technol, 1977, pp. 737-748.
 98. Udupa, S., "Collision Detection and Avoidance in Computer Controlled Manipulators", Ph. D. Dissertation, Dep. Elec. Eng., California Inst. Technol., 1977.
 99. Varshavskii, V.I. and Vorontsova, I.P., "On the Behaviour of Stochastic Automata with Variable Structure", Automat. Telemekh., Vol. 24, March, 1963, pp. 353-360.
 100. Wang, S.H., "Computed Reference Error Adjustment Technique (Create) for the Control of Robot Manipulators", 22nd Annual Allerton Conf. on Communication Control and Computing, October 1984.
 101. Wang, S.H., and Horowitz, I., "Create a New Adaptive Technique ", Proc. of the Nineteenth Annual Conf. on Information Sciences and Systems, March 1985.
 102. Winston, P.H., and Brown, R.H., "Artificial Intelligence: An MIT Perspective", Vol. 2, the MIT-Press, Cambridge, Massachusetts, and London, England, 1983.
 103. Whitney, D., "Resolved Motion Rate Control of Manipulators and Human Prostheses", IEEE Trans. on Man Mach. Syst., MMS-10, 1969, pp. 47-53 .
 104. Wu, C., "A Kinematic CAD tool for the design and Control of a Robot Manipulator", Int. Journal on Robotics Research, 1984, Vol. 3, pp. 58-67.

END

08.08.88

FIN