Neuro-Fuzzy Motion Planning for Robotic Manipulators

Kaspar Althoefer

Thesis submitted for the Degree of Doctor of Philosophy
Department of Electronic and Electrical Engineering
King’s College London, University of London

1996

Abstract

On-going research efforts in robotics aim at providing mechanical systems, such as robotic manipulators and mobile robots, with more intelligence so that they can operate autonomously. Advancing in this direction, this thesis proposes and investigates novel manipulator path planning and navigation techniques which have their roots in the field of neural networks and fuzzy logic.

Path planning in the configuration space makes necessary a transformation of the workspace into a configuration space. A radial-basis-function neural network is proposed to construct the configuration space by repeatedly mapping individual workspace obstacle points into so-called C-space patterns. The method is extended to compute the transformation for planar manipulators with n links as well as for manipulators with revolute and prismatic joints.

A neural-network-based implementation of a computer emulated resistive grid is described and investigated. The grid, which is a collection of nodes laterally connected by weights, carries out global path planning in the manipulator's configuration space. In response to a specific obstacle constellation, the grid generates an activity distribution whose gradient can be exploited to construct collision-free paths. A novel update algorithm, the To&Fro algorithm, which rapidly spreads the activity distribution over the nodes is proposed. Extensions to the basic grid technique are presented.

A novel fuzzy-based system, the fuzzy navigator, is proposed to solve the navigation and obstacle avoidance problem for robotic manipulators. The presented system is divided into separate fuzzy units which individually control each manipulator link. The competing functions of goal following and obstacle avoidance are combined in each unit providing an intelligent behaviour. An on-line reinforcement learning method is introduced which adapts the performance of the fuzzy units continuously to any changes in the environment.

All above methods have been tested in different environments on simulated manipulators as well as on a physical manipulator. The results proved these methods to be feasible for real-world applications.

Acknowledgements

Most projects involve the collaboration of people, composing this thesis has been no exception.

I am thankful to David Fraser for his supervision. I appreciate the lively discussions we had in which he supplied me with academic arguments that have aided the understanding of my research. I value very highly the fact that he was always available to answer questions. This thesis has benefited a great deal from his advice.

The suggestions and comments of Guido Bugmann have been a source of constant reference and inspiration. For the guidance he provided whenever I tended to lose the path I owe him more than I can express in words.

Robert Schwann patiently listened to many of my half-developed ideas and provided useful advice to overcome many hurdles. His contributions are noted with sincere thanks.

I must also express my gratitude to Ton Coolen, Bart Krekelberg and Rasmus Strange Petersen for their invaluable comments and suggestions with respect to questions on the training of fuzzy controllers and the dynamics of Hopfield networks. Moreover, I am grateful to Mark Plumbley for his contributions to the work on the workspace to C-space transformation. I also would like to thank Rahul Jaitly for the contributions during our collaboration on the vision-based workspace to C-space transformation. Thanks also go to Henning Schmidt who gave lots of helpful comments on how to improve the description of the fuzzy navigator. Furthermore, I have to thank Panos Zavlangas for the extremely successful implementation of the fuzzy navigator. I also would like to thank Andreas Wanitschek for proof-reading crucial parts of this document.

Thanks are also due to Trevor Clarkson, Tony Davies, Mark Sandler who watched my progress with a vigilant eye and always had an ear for the questions I came up with during my time at the Department of Electronic and Electrical Engineering.

The list of acknowledgements would be incomplete, if I did not mention my gratitude to the Erasmus students Laurent Alluzen, Jörn Behrenroth, Marcello Fini, Gabriele Landi, and the many colleagues in the Department, especially Ebi Agboraw, Jamil Ahmad, Shabbir Bashar, Gabriela Castellano, Chris Christodoulou, Jane Croucher, Vassilios Dionissopoulos, Terhi Garner, Tareq Hamdi, Tarig Hanif, Panos Kudumakis, Nilceu Marana, Allan Paul, Georgi Petkov, Aubrey Sandman, Atif Sharaf, Fei Xia, Chengping Xu and Jia Hong Yin who provided me with an environment which was enjoyable to work in. Thanks also for advice and technical support by Peter di Cara, Terry Coleman, Peter King, Talat Malik, Mustaq Mohammed and Maisie Payton.

My research was facilitated through the generosity of the Dr. Jost-Henkel- Stiftung, Düsseldorf, Germany which supported me financially during 1993-1996.

My brother and my sister shared with me more than I could possibly express here and were always there when I needed them. I also would like to thank my parents for their unfailing financial support and endless moral encouragement.

Lito Apostolakou did not only help immensely to bring this thesis into its final shape but also bravely endured my mood swings during this stressful time. Without her continuous support this work would never have been possible.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

A variety of definitions of the notions "robot" and "motion planning" can be found in literature (see for example [Lee96 and references therein, Gupta92, Ralli94, Latombe91]). In general terms, a robot is considered to comprise an intelligent, versatile mechanical structure or body whose movements are steered by a controlling apparatus (nowadays usually a computer). Robots are expected to accomplish their tasks in a real world, a fact that brings about the important problem of motion planning which takes into account the inevitable interactions between the robot and objects in its environment. Basically, motion planning is a process which is concerned with finding a sequence of configurations from a start to a goal configuration, thereby avoiding collisions with objects in the robot's workspace.

Motion planning for robotic manipulators is a research field which has long fascinated researchers from a diversity of backgrounds [Latombe91]. Mechanical and electrical engineers, computer scientists, mathematicians, physicists, and more recently, neural networkers, experts in fuzzy logic, biologists and brain scientists have been developing algorithms and methods with the aim to create intelligent and autonomous robots. However, so far emphasis has been mainly placed on theoretical and simulation-based studies, while real-world research work is still in its infancy. The evaluation of developed methods in real-world experiments is often neglected by scientists, leaving a niche for the more practically oriented researcher.

Robotic motion planning can be divided into two areas: global path planning, and local navigation.

Global path planning usually makes use of a state space or map which, for example, describes valid and forbidden manipulator configurations (see [Lozano83, Latombe91] and Section 1.2.2). A path planner is then concerned with the searching of a path inside the region of valid configurations, thereby connecting a start state with a goal state. A global planner is one which stops searching when either a path is found or it is shown that no such path exists (see Section 1.2.3).

While global path planning strategies carry out their computation prior to the robot movement, local navigation techniques commonly work on-line. The navigator is programmed to compute an actuation command in reaction to the information acquired by external sensors viewing the environment in the manipulator's vicinity. Usually, a navigator provides reactive behaviours, such as "avoid obstacles" and "reach target". Due to their very short response time, local navigation techniques are commonly used to steer robots in dynamic or uncertain environments (see [Lee96, Brooks86] and Section 1.2.4).

This thesis proposes and investigates novel motion planning strategies for robotic manipulators which fall within the areas of global path planning, and local navigation. This document is structured into five chapters.

Chapter 1 - Introduction

The sections of this introductory chapter describe the methodology of the thesis and, through an overview of research work conducted in the field of robot motion planning, place the thesis in context. The introduction concludes with a summary of contributions made.

Chapter 2 - Workspace to C-space Transformation

Chapter 2 presents a novel technique based on a *radial-basis-function* (*RBF*) neural network to compute the *configuration space* (*C-space*) of manipulators. This space which contains the valid and forbidden manipulator configurations can serve as an input to a global planning strategy. The neural network is trained to react with a so-called *C-space pattern* to an obstacle point in workspace. Multiple accesses of the network allow the rapid construction of the complete C-space. Experiments with real and simulated manipulators are presented.

Chapter 3 - A Neural Resistive Grid for Path Planning

Chapter 3 introduces a global path planning strategy which is based on a neural implementation of a computer-emulated resistive grid. A new update method which is particularly well suited for manipulator path planning is presented and compared to other planning algorithms. Extensions of the original strategy include planning in non-topological maps and the incorporation of a “soft” safety margin. The strategy has been applied to different manipulators in a range of real and simulated environments.

Chapter 4 - Fuzzy-Based Navigation and Obstacle Avoidance for Robotic Manipulators

Chapter 4 proposes a fuzzy-based *navigator* for robotic manipulators. This reactive system combines goal directed behaviour with obstacle-avoidance behaviour to manoeuvre a manipulator in static as well as dynamic environments. The rule base of the navigator has been constructed using different methods - common sense, trial and error, and learning rules. The navigator’s implementation and functioning is described and demonstrated by means of simulations, and real-world experiments.

Chapter 5 - Conclusions and Future Work

Chapter 5 summarises and evaluates the results of this research work and, moreover, suggests directions for further research.

1.1 Methodology

This thesis puts great emphasis on the development of algorithms which can be applied to physical manipulators. Most of the techniques and strategies developed here have been successfully tested on real manipulators.

The experimental robot system used was the MA 2000 manipulator[1]. The original control unit had been replaced by a transputer network consisting of a number of transputers arranged on separate boards [Graham90], and hosted by an IBM-compatible PC. One of the boards, called the port, has been specifically designed to communicate with the manipulator [Higgins88]. The port is connected via a two-directional interface with the manipulator. The network transmits pulse-width-modulated control signals to the interface which amplifies those signals and feeds them into the joint motors. The interface receives the analogue joint potentiometer values, converts them into digital ones, and sends the latter to the port of the transputer network. Since the user of the system can directly control the manipulator’s motors and read the joint configurations, routines can be developed to do low level control [Azhar93, Fraser93] as well as path planning [Althoefer95d, Althoefer95e]. Details on the control

[1] The MA 2000 manipulator has been originally developed by TecQuipment Ltd. for Open University courses in manufacturing and robotics.

interface and the communication software in the port-transputer can be found in the appendix of [Althoefer94b].

The path planning experiments, as described in Chapters 2 and 3, have been carried out off-line on an IBM-compatible PC. The image of the workspace was acquired by a CCD-camera which was connected to a specialised image grabbing board [Jaitly96c]. The resulting trajectory has been then downloaded to the transputer network which steered the manipulator (for results, see Chapters 2 and 3). The real-world experiments concerning the fuzzy-navigation have been carried out entirely on the transputer network [Zavlangas96].

Notwithstanding the importance of real-world experiments, the advantages of simulations should not be underestimated. New algorithms can be tested in a virtual environment prior to the real test thereby avoiding unnecessary damage of expensive equipment and often speeding up the development process. For example, the use of a simulator was essential for the training of the fuzzy navigator. At the start of the training, the navigator's rule base was initialised with random values, and the manipulator performed arbitrary movements which resulted in "virtual" collisions with the simulated obstacles. Reinforcement learning rules were formulated in such a way that the navigator's commands quickly adapted to the environment and provided collision-free manipulator movements (see Chapter 4).

Simulation experiments have been carried on an IBM-compatible PC using the software package MATLAB, Version 4.0.[2] Those simulations were on models of the MA 2000 as well as on manipulators with "stick-like" links with zero-width. MATLAB is a powerful signal processing tool which enables researchers quickly to convert their ideas into executable programs and to visualise the results employing the advanced graphic routines. Moreover, real-world data can be imported and computation results can be exported. Both these features have been often utilised in this research work. This meant that data recorded from the MA 2000 were incorporated into the simulations, while the results of the simulations regarding the MA 2000 were depicted graphically, and applied without further modifications to the real manipulator system. The MATLAB-add-on package, Simulink as well as the Signal-Processing-WorkSystem by Alta Group of CADENCE Design Systems - both block-oriented programming tools - were valuable to get acquainted with neural network and fuzzy paradigms (see also [Althoefer93, Althoefer94a]).

The project concerning the image-based workspace to C-space transformation described at the end of Chapter 2 was the result of a collaboration with R. Jaitly (Dept. of Electronic and Electrical Eng., King's College London). The neuro-resistive grid which has been thoroughly investigated and further developed as described in Chapter 3 is based on work of and collaboration with G. Bugmann (School of Computing, University of Plymouth). The real world implementation of the fuzzy navigator (Chapter 4) was carried out during an MSc-project under D. Fraser's (Dept. of Electronic and Electrical Eng., King's College London) and the author's supervision. The MSc-student was P. Zavlangas (Dept. of Mechanical Eng., King's College London). The manipulator system had been improved by F. Azhar as well as T. Higgins (both Dept. of Electronic and Electrical Eng., King's College London).

[2] MATLAB is produced by The Mathworks Inc.

1.2 Constructing the C-space, Global Path Planning, Local Navigation: An Overview

1.2.1 Manipulator Motion Planning

Articulated machines have revolutionised industrial processes, especially in the manufacturing sector. They are known for their indefatigable effort to assemble mass products, such as cars or electronic equipment. Strictly speaking, these machines are not considered robots, since they are not intelligent. Generally, they accomplish a task in a structured environment by executing a pre-recorded sequence of motions that have been previously taught by a human operator (see Chapter 3). With the need to increase the flexibility of mass production equipment in order to be able to adapt more quickly to changing consumer markets, but also with the desire to open up new application areas, research pursues the goal to construct robots which ultimately carry out specific tasks without further human interventions.

At present, one of the main areas research is focusing on is concerned with the operation of robots in terrains where human access is restricted, mainly because of health risks. Telerobotic is a research field which becomes increasingly important. New application areas are hazardous environments such as nuclear decommissioning (for example in a nuclear plant [Racz92]), manipulation of toxic chemicals or contagious medical samples, operation of toxic waste tips, land mine disposal, painting tasks, subsea exploration, and space experiments [Ralli96]). Telerobotic has experienced a tremendous boom with the development of the Internet which supports the communication between computers world-wide. Some initial installations running at the moment allows any Internet user to operate a camera equipped mobile robot [http://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/Xavier/], the manipulator TeleRobot at the University of Western Australia [http://telerobot.mech. uwa.edu.au/cgi-win/telerobt.exe], a robotic telescope at Bradford, UK [http://www.eia.brad.ac.uk/eia.html], a remote live telerobotic camera in Almeda California, USA [http://citynight.com/camera], etc. The user of a World-Wide-Web browser can send control commands to position the telerobot and usually receives imagery data acquired by a camera mounted on the robot. Since the user's commands usually are high level commands path planning and sensory guided obstacle avoidance plays an important role in those Internet experiments.

Path planning also becomes increasingly useful in areas, such as virtual reality [Klawsky93], and computer controlled games.

Another field which is increasingly attracting interest from researchers is concerned with the investigation of the close collaboration of humans and robots. In particular, the application of robots in the domestic sector (for example cleaning services, support of the disabled, delivery and distribution services) is gaining ground. In these applications, a very precise motion [Morasso95] is often not necessary as a more coarse planning system which provides a rough plan prior to the robot's motion is generally sufficient. The computed motion is adjusted to the environmental changes which are usually provoked by a human. Increasing concern about safety issues has prompted the emergence of safety engineering [Redmill95, Zicky95], and recent efforts in this field aim at proving the reliability and predictability of neural networks, fuzzy controllers and expert systems in safety critical applications [Morgan95, Helliwell95, Johnson93].

1.2.2 The Computation of the C-space and the Building of Maps

Many path planning strategies developed so far make use of the robot's configuration space, a space where the robot is represented by a particle [Latombe91 and references therein, Althoefer95d, Althoefer95e, Bugmann95, Ralli96, Glasius94, Connolly90, Tarassenko91]. To employ the configuration space concept in a real-world path planning task mapping the geometry of the task into the configuration space becomes necessary [Latombe91]. This transformation process has been also denoted *FindSpace* [Lozano87]. A variety of methods has been developed over the last two decades to accomplish this often highly non-linear transformation of the workspace obstacles into the C-space counterpart [Lozano83, Lozano87, Latombe91, Althoefer95c, Newman91, Branicky90, Maciejewski93, Gupta92]. Despite their diversity, these methods have one thing in common: the C-space is a discretised map or a graph whose nodes represent the manipulator's configurations. Map nodes which represent collision-free robot configurations contain a different value than those nodes which represent collisions. Some approaches use a multi-valued logic incorporating a description of configurations which are uncertain [Latombe91, Siemiatkowska94b]. Those maps can be metric maps (as used in most experiments of this thesis) or topological maps [Lee96]. It has been reported that solving the path planning problem in its generality is almost intractable (see [Ralli94]). Every planning strategy has to be adapted to the particular problem. However, employing the configuration space generates some kind of generalisation, since any strategy which solves the planning problem for a point robot can be utilised in such a space.

There is a great difference between the approach to constructing a map for mobile robots and to building a map for manipulators. Especially recently, explorative methods inspired mainly by animal research have been found to be a good means of generating such maps representing the environment of mobile robots [Sutton90, Lee96, Zimmer94]. These methods assume that the environment is to a great extent static and the map is still valid when the robot returns to an area visited earlier on. To be able to perform some movements while moving in unknown terrain, the mobile agent is additionally equipped with a behavioural-based control mechanism which supplies commands, like goal-directed commands, wall-following commands, obstacle avoidance and similar (see for example [Lee96, Tschichold96]). Once the map is built, the robot can invoke a planning strategy to find a path from its current position to a goal position. One of the problems with this approach is that it is often difficult to decide when to switch from one strategy to another [Lee96]. An alternative approach models robot and environment. Instead of moving the physical robot, exploration and map building takes place in the simulated world. The latter approach tends to be faster, but of course, presumes that a model exists. Combinations of the two aforementioned approaches have been also proposed (see for example [Sutton90]).

However, the exploration of the robot's workspace is a very time-consuming process. Path planning can be performed reliably only if obstacles do not move after the map has been built. In an extreme case, a moved obstacle may close the gap through which the planner proposed a path, and another lengthy exploration phase may be required to construct a new path. Most useful, as reported in [Lee96], is the combination of planning in the map and a reactive behaviour with emphasis on the latter. Explorative strategies for manipulators are only sensible in static environments [Ritter92] or useful in solving docking problems which require fine movements that usually cannot be computed in discrete maps (see for example [Rucci93]).

Intelligent manipulators are expected to manoeuvre in possibly changing environments. Preferably, the time to execute tasks should not be drastically prolonged by the planning program. It is an advantage that the manipulator's workspace is confined to a relatively small area, if compared to the possibly very huge workspace of a mobile robot. A representation of this kind of workspace can be acquired for example by a camera or a multi-camera system. From this information a C-space can be calculated [Lozano87, Jaitly96b].

Traditionally, the configuration space of manipulators is computed algebraically. In order to facilitate the calculation of intersections between obstacles and manipulator links, all objects involved are represented by simple polygonal or polyhedral objects (see for example [Lozano87, Gupta92, Latombe91, Hwang90, Winkelmann96]). The constructed discrete C-space is made up of cells for each joint denoting valid and forbidden configurations.

An alternative purely numerical approach was proposed by Newman *et al.* [Newman91]. Their method is based on the transformation of obstacle points in a discrete workspace representation (for example the pixels in bitmap image acquired by a camera) into a discrete C-space pattern. This method exploits the symmetry inherent to most modern manipulator types (see also Chapter 2). Multiple computations allow the fast mapping of complex obstacles. The advantage of the numerical approach over the analytical method is that the C-space patterns can be constructed in such a way that they describe in exact terms the C-space region which corresponds to all collisions between the manipulator in use and an obstacle point. Any of these C-space patterns which are stored in a look-up table is accessed on the occurrence of a point obstacle in workspace [Newman91]. The original method has been extended by Althoefer *et al.* to allow the C-space construction for redundant planar manipulators with a high degree of freedom (DOF) for manipulators with revolute as well as prismatic joints, and for manipulators which comprise a combination of the two joint types [Althoefer94, Althoefer95, Althoefer95b, Althoefer95c]. The storage of the patterns in a neural network proved to need little memory (see [Althoefer95c] and Chapter 2).

Recently published research suggests the use of Kohonen networks (also called self-organising feature maps) for robotic applications. Ritter *et al.* suggest an extension of the Kohonen network which learns the inverse kinematics of a robotic manipulator just from the output of cameras which observe the manipulator moving arbitrarily in workspace (see [Ritter92] and see also [Kuperstein91, Mel95] for other neural network architectures which learn the inverse kinematics of a manipulator by example). The network by Ritter *et al.* needs many thousand training cycles until it successfully associates the positions of the manipulator's end effector with the appropriate joint configurations. Their work concerns itself with simulated manipulators only. Further work has been proposed by Zimmer *et al.* [Zimmer94, Zimmer94c]. Their work is mainly related to the constructing of maps for mobile robots. An explorative phase is needed to build the map which automatically expands when new regions are discovered. This method also suffers from long training cycles.

1.2.3 Global Path Planning in C-space

Having reviewed the work concerned with the computation of a configuration space, this section presents an overview of planning methods in the configuration space, also denoted *FindPath* [Lozano83].

The most popular methods which make use of a robot's configuration space are road map [Latombe91] and cell decomposition methods [Gouzenes84]. In the road map technique a list of all possible paths between the given obstacles is generated and stored. After this list or road map is created, it can be used as a set of the standardised paths. The path planning procedure in this case entails choosing a subset of concatenated paths from the list which connect start and goal configuration. The visibility graph method is based on road maps and usually applied to two-dimensional spaces where obstacles are represented by convex polygons. The list contains all straight line connections between vertices of obstacles which do not intersect with any obstacle. An extension of the visibility graph to 3D has been proposed recently [Jiang94]. His method finds the shortest path, if any such exists, in 3D-configuration space with convex polyhedral obstacles. Any found path leads along the edges of obstacles, i.e., it is semi-free and therefore not safe for some implementations. The use of this method for manipulators with three degrees of freedom is restricted owing to the fact that C-space obstacles of manipulators have complex shapes which cannot be easily described by polyhedral forms (see Chapter 2). Other road map methods are: Voronoi diagram, freeway net and silhouette method (see [Latombe91] for an overview).

Cell decomposition methods decompose the free area of the configuration space into small cells located between the obstacles. The cells are made so small that path finding inside the cells can be contrived with a simple algorithm (usually along a straight line). The cells of the free space are placed as nodes in a graph (connectivity graph) which represents the adjacency relation between the cells [Latombe91]. Two nodes are connected via a link if the link crosses one cell or one boundary. From the graph, a sequence can be easily extracted which describes a collision-free path between start and goal configuration. Cell decomposition methods can be either exact or approximate. Employing exact cell decomposition methods, the free space is divided into cells of different shapes. The union of these cells covers the free space completely. Two cells are adjacent if they share a boundary. Connectivity graphs are usually constructed through the centre points of the shared boundaries. A path or channel can be constructed along the stored graph nodes. Approximate cell decomposition is a numerical method which produces cells of defined shape (for example rectangloids) whose union is strictly included in the free space. A configuration space is initially bounded by a rectangloid. In case of a two-dimensional C-space, the rectangle is recursively divided into four rectangles. This type of decomposition is called quad-tree because it can be represented by a tree of degree four (oct-trees are used for three-dimensional configuration spaces). Decomposition continues until a path can be constructed through cells which lie entirely in the free space. Usually, approximate cell decomposition methods are implemented in a hierarchical way. At first the space is divided by means of a rather coarse resolution which is then increased until a path is found or a resolution limit is attained [Latombe91]. Being not sensitive to obstacle shapes, the approximate cell decomposition can be applied to a manipulator's C-space. However, its computational complexity is sensitive to the obstacle distribution; a crowded obstacle distribution may be especially computational expensive. It has been reported that the cell decomposition methods are only tractable for robots with small degrees of freedom [Barraquand91].

The search for a path in the graph or tree generated by above methods is usually carried out by employing a graph search method, such as the A*-algorithm. The neuro-resistive grid presented in Chapter 3 can also be used to search such a graph.

Latombe *et al.* proposed numerical potential field techniques [Latombe91, Barraquand92] and the "distributed representation approach" [Barraquand91] which make use of the robot's workspace and its configuration space. Since the basic potential field approach [Khatib86] on which their work is based can get stuck in local minima, Latombe *et al.* added some costly strategies (for example random search techniques) which turned the algorithm into a global one. Although, the algorithm is able to handle robots with many degrees of freedom, their planners seem sometimes to overlook the most obvious and seemingly shortest route towards the goal. Their planners are very complex and specifically adapted to particular robots [Latombe91]. Their published research results refer to line robots only.

Recently, the use of discrete harmonic functions have been suggested for solving the path planning problem for robots and manipulators [Conolly90, Connolly93, Tarassenko91, Bugmann95, Althoefer95e, Kim91]. These functions overcome the main disadvantage of potential field methods by providing solutions with one unique solution only. Resistive grids, those emulated on computers [Conolly90, Bugmann95, Althoefer95e] as well as those implemented in hardware [Marshall94, Tarassenko91] can be employed to compute solutions to harmonic functions (see also Chapter 3). The analytical computation of harmonic functions has been studied by Tarassenko *et al.* [Tarassenko91]. This work is only applicable to very simplified (for example circular) obstacles and, therefore, not useful to construct a path among the highly-irregular shaped C-space obstacles of manipulators (see also [Wolf68]). Especially the hardware implementations of a resistive grid is expected to provide a tremendous speed up in planning time [Marshall94, Tarassenko91].

Chapter 3 of this thesis focuses on aspects of path planning in a resistive grid. A variety of planning strategies based on discrete maps have been developed by other researchers which prove to be very similar to the grid-based approach. Those strategies can be described as extensions of the dynamic programming method proposed by Bellman (see [Bellman57] and Chapter 3). One offshoot commonly employed in robotics is the A*-algorithm [Latombe91 and references therein]. Further path planning methods with similar properties are called: Hopfield-type networks [Glasius94, Kassim95], cellular neural networks [Siemiatkowska94b], numerical potential field techniques [Ralli94], distance transforms [Lee96, Boult90, Mckerrow91]. Different update methods are employed to generate an activity distribution over the map. The gradient of this distribution can be exploited to construct a path from a start configuration to a goal configuration. All these techniques are resolution complete, that is, a path is found, if one exists in the discrete space (for details see Chapter 3).

Path planning strategies which make use of a configuration space representation have been criticised for their complexity and exorbitant memory requirements for high-dimensional robots. Recent papers, however, show that C-space-based path planners can be used to effectively plan paths for 6 DOF-manipulators on single-processor computers [Ralli94]. With the on-going developments in the area of parallel computers, further improvements in terms of transformation speed are expected. Parallel processing is especially suitable for the transformation technique proposed in Chapter 2 of this thesis.

1.2.4 Local Navigation

Ideally, a robot should perform its tasks quickly and the speed of its moving parts should be only limited by the robot's inherent properties (like torque of actuators, friction in joints and gears, weight and inertia, etc.). This challenging aim can (if at all) be reached in an environment where not only the initial locations of obstacles are precisely known, but also their locations versus time. This is rarely the case in non-industrial applications. On-line navigation and obstacle avoidance techniques represent an interesting approach to tackle this problem. These techniques can deal with moving obstacles as well as uncertain environments.

Khatib pioneered the artificial potential field methods implementing a collision avoidance algorithm for mobile robots and manipulators operating within a dynamic environment [Khatib86]. This method has been mainly employed as an on-line method for local navigation and obstacle avoidance, but has been also influential to planning [Latombe91]. For example, offsprings of this method are the path planning techniques designed by Latombe *et al.* (see previous section and [Barraquand92]). Khatib's algorithm essentially uses two potential fields which are superimposed: the first field surrounds the target and its gradient can be interpreted as an attractive force on a point-sized robot, while the second field which is active in the vicinity of obstacles exerts a repelling force on the point. Superimposing the two fields creates a potential distribution (not to be confused with the potential distribution in a resistive grid) which can be employed for moving a robot without collision towards a goal state when the geometry of the problem does not contain local minima [Khatib86, Latombe91].

Brooks initiated the research in the field of behaviour-based or reactive navigation [Brooks86]. His well known navigation system is called subsumption architecture. The main principles of his work (which should be employed to produce simple, robust and cheap robots, as he has said) are a collection of modules which are interconnected on different layers with different hierarchies. These modules are for example wall following, obstacle avoidance, goal reaching, etc. Depending on sensory input, a module becomes active and generates a command for the robot. His work and the one of his successors is almost exclusively applied to mobile robots (see for example [Handelman90]).

Fuzzy-based obstacle avoidance and navigation for mobile robots can be seen as an extension of Khatib's and Brook's work [Althoefer96, Reignier93, Tschichold96]. In a fuzzy-based robot navigator, rules such as those suggested by Brooks are integrated into the navigator's rule base. The similarity to Khatib's approach becomes clear when one considers that some of these rules exert an attracting influence on the robot, while others exert a repelling influence.

While Brooks' system resembles an expert system where for any input signal one specific reaction module or a specific combination of modules is active, the fuzzy approach is a parallel processing approach and each input contributes to the final decision [Kosko92, Tschichold96, Althoefer96]. Additionally, owing to the fuzzyfication stage inherent to all fuzzy controllers, a mode of approximate reasoning is introduced which allows the controller to deal with vague and incomplete input data. This becomes particularly important when dealing with imprecise information from sensors [Hoffmann96, Lee96].

Neural network learning methods ([Reignier94, Kosko92] and Chapter 4)) and genetic algorithms [Hoffmann96] have been adapted to train the parameters of fuzzy navigators. In contrast to neural network approaches, including those used for robot navigation [Sharkey96], the rule base of a fuzzy controller is interpretable. Two main advantages follow from this. Firstly, training can start from a rule base which has been set up by common sense rules. This reduces training time and provides a reasonably good behaviour of the controlled system already at the beginning of the training. The latter aspect is important, if training is to be carried out on a real-world system, since any damage of equipment should be avoided. Secondly, the rule base is humanly understandable even after training and can be further modified if necessary.

Reports on recent advances in fuzzy and neural-network based navigation and obstacle avoidance for mobile robots can be found in the following publications [Reignier93, Tschichold96, Roth93, Maier95, Song92, Maties94, Hoffmann96, Skubic93, Mitchell96].

The disadvantage of local navigation methods in general is the occurrence of local minima, a fact which occasionally causes them to fail to reach the goal configuration [Millan92]. One possibility of overcoming this problem is to construct a hybrid system where a path planner and a navigator are combined [Latombe91, Tschichold96]. For example, a global planning strategy provides off-line a coarse trajectory prior to the executed motion, while on-line a local navigation is employed to refine this trajectory and to adapt to a changing and uncertain environment. Another approach is to initially attempt to find a path using a local method and to switch to a more global strategy only in cases where this method fails.

1.3 Contributions made by this Thesis

This thesis describes an experimental investigation into the complementary issues of global path planning and local navigation for robotic manipulators. The contributions of this research are:

- The development of a neural-network-based workspace to configuration space transformation technique for robotic manipulators with revolute joints; the extension of the technique to compute the transformation for planar manipulators with *n* links as well as for manipulators which with revolute *and* prismatic joints;
- The development of an effective update method which rapidly generates an activity distribution in a computer emulated resistive grid in order to plan paths in the configuration space of manipulators;
- The design of a novel fuzzy-based navigation system for robotic manipulators;
- Application of above methods to a physical manipulator and thorough evaluation of their real-world feasibility in different environments.

Chapter 2

Workspace to C-space Transformation

This chapter proposes a novel neural network architecture based on asymmetric second-order B-spline functions to rapidly transform the representation of a manipulator's workspace into a *configuration space* (*C-space*) representation. This transformation becomes necessary in real-world path planning applications which employ the manipulator's configuration space. The C-space of a manipulator is usually the space spanned by the arm's joint parameters. Any point in this space represents a configuration which either denotes a collision between the manipulator and a workspace obstacle or is collision-free. Many path planning strategies developed up to today make use of the C-Space, since the motion planning for the complex manipulator structure is transformed into the planning for a single point in C-space. (Figure 2.4-1 and Figure 2.4-2 show a typical C-space for a single point.)

The *B-spline functions* network, which is a offshoot of the family of *radial-basis-function* (*RBF*) networks, transforms obstacle primitives occurring in the robot's workspace representation into a corresponding C-space representation, called *C-space pattern*. The focus of this chapter is on the transformation of the most fundamental primitive, the *point obstacle*. The complete configuration space representing all valid and all forbidden configurations can be obtained by repeatedly computing C-space patterns for these elementary obstacle points. The neural network is trained to recognise those obstacle primitives and to respond with the associated C-space patterns. During training the network's output is compared to new, unknown patterns, and wherever the interpolation error is too high, the network structure is expanded by new nodes. The interpolating features of the B-spline function network are exploited to achieve a good approximation for untrained patterns.

This chapter is organised as follows: Section 2.1 gives an introductory description of the workspace to configuration space transformation technique. An overview of related work in Section 2.2 is followed by a description of the general aspects of the configuration space of robotic manipulators in Section 2.3. Section 2.4 describes the mapping of obstacle primitives situated in the workspace of two-link manipulators into a configuration space representation. In Section 2.5 the technique is expanded to include manipulators with *n* links. A neural network implementation is presented in Section 2.6. A simulation of a three-dimensional manipulator is presented in Section 2.7. The application of the network-based technique to real-world data recorded from an experimental manipulator is described in Section 2.8. A summary of this chapter is given in Section 2.9

2.1 Introduction

Moving a manipulator from a start configuration to a goal configuration in a *workspace*[1] cluttered with obstacles demands path planning. Path planning can be carried out effectively in the *configuration space* (*C-space*). The C-space of a manipulator is generally the space described by the variable joint parameters, such as the rotation angle or slide position. The C-space is divided into two subspaces: the *C-obstacle region* and the *C-free region*. The C-obstacle region represents the constraints imposed on any part of the manipulator

[1] The workspace is the description of the area which can be reached by any part of the manipulator.

[Newman91, Latombe91]. This region which can be regarded as a forbidden region describes collisions between the robot and obstacles as well as collisions between parts of the robot. The C-free region is the region in which the robot can navigate safely. In C-space, each of the robot's configurations is represented as a single point, thus, the path planning for the complex structure of a robot is transformed into the planning for a point [Lozano83, Lozano87, Latombe91, Branicky90, Newman91, Maciejewski93, Fox92, Fox94]. Many path planning and obstacle avoidance techniques which make use of the C-space have been developed (For example see [Latombe91, Barraquand91, Bugmann95, Althoefer95e, Ralli94] & Chapter 3).

Any real-world path planning system that makes use of the configuration space depends inevitably on a transformation process which maps the workspace representation, usually acquired by sensors (such as range finders or camera systems [Indyk94, Jaitly96b]) into a C-space representation. The overall planning time of such a path planning system depends not only on the time spent on planning, but also on the time spent on transforming the workspace into the configuration space. The configuration space transformation can be a time consuming process [Newman91, Lozano87, Latombe91]. To perform path planning in a real-time application, a fast and efficient transformation method becomes necessary (see Section 2.6.1).

This chapter presents a technique by which a workspace representation is rapidly transformed into its C-space counterpart. This technique is based on the most fundamental C-space transformation which originates from avoiding a workspace obstacle point [Branicky90]. The *C-space obstacle* (see Section 2.4.2) that corresponds to a workspace obstacle point is called here a *C-space pattern*. An obstacle in a workspace can be described by the union of a set of points. The corresponding C-space obstacle is the union of the C-space patterns corresponding to all those obstacle points. Thus, multiple point transformations allow the transformation of any workspace obstacle into the corresponding C-space obstacle [Newman91].

The transformation of workspace obstacle points into C-space patterns is highly nonlinear for manipulators with *revolute* joints[2] [Althoefer95, Althoefer95c]. This chapter focuses on the obstacle point transformation for two-link revolute robot arms. This basic arm type is part of commonly used industrial manipulators (see [Newman91] and Figure 2.1-1 (middle) and (right)) and can be expanded to compute the C-space patterns for redundant manipulators with *n* planar links (see Figure 2.1-1 (right) and Section 2.5). Due to the symmetrical properties of such manipulators, any obstacle point on a circle around the manipulator's base causes a C-space pattern with identical shape. Thus, the shape of C-space patterns depends only on one feature: the distance between the manipulator's base and the obstacle point. Here, each such C-space pattern is discretised into a set of vectors where each vector denotes a forbidden configuration in C-space (Section 2.5).

Section 2.6 of this chapter proposes a neural network architecture to transform workspace obstacle points into distinct C-space patterns [Althoefer95, Althoefer95b, Althoefer95c]. This network architecture is a *radial-basis-function* (RBF) neural network [Brown94, Bishop95, Bose96]. Instead of using Gaussian functions in the "*receptors*" of this network type as commonly done, the network here employs triangular (*second order B-spline*) functions which allow a piece-wise linear interpolation of the input space (see Section 2.6 and [Brown94, Althoefer95c]). The neural network receives at its input a stimulus which represents the distance between the manipulator and the obstacle point in workspace. The network is trained to output a discretised C-space pattern in response to such an input

[2] Revolute joints represent a rotatory connection between two manipulator links.

stimulus. A newly developed training procedure adapts the network size to the underlying problem (see Section 2.6.2 and [Althoefer95c]). The RBF network interpolates smoothly for those input points which are not part of the stored data set [Althoefer95, Althoefer95c, Brown94].

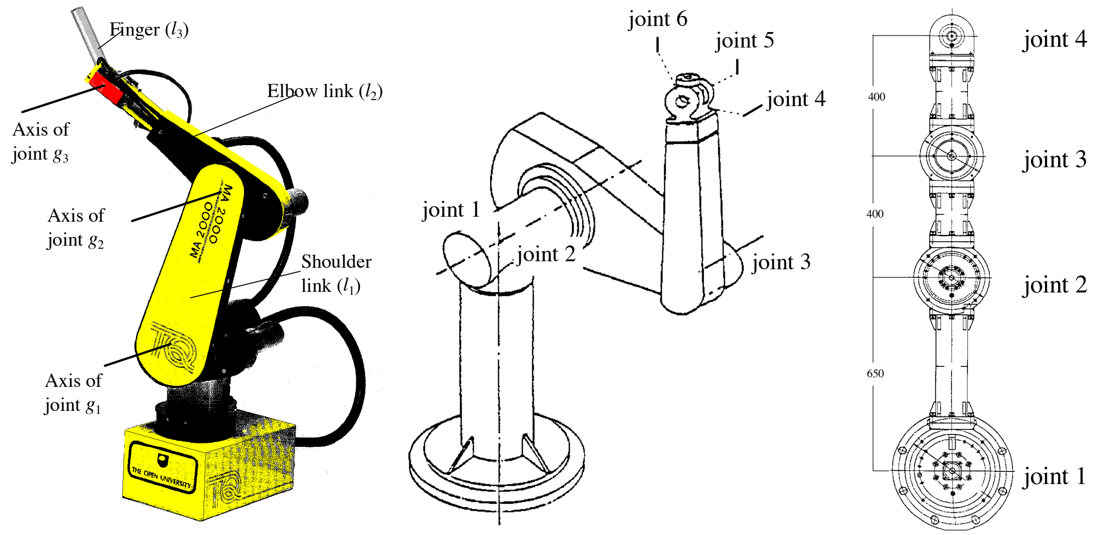


Figure 2.1-1: (left) The MA 2000 manipulator; (middle) The Puma manipulator (from [Chien95]); (right) Redundant SCARA robot RedRob (from [Risse95]).

The feasibility of the proposed technique is demonstrated by considering the workspace to C-space transformation for the experimental manipulator, MA 2000 (see Figure 2.1-1 (left) and Appendix A). The MA 2000 has a kinematic structure which is similar to the structure of many standard industrial manipulators, such as Unimation Puma robots (see Figure 2.1-1 (middle)). In particular, the B-spline functions network has been used to calculate the configuration space of the shoulder and elbow link of the MA 2000 (Section 2.8.1). Furthermore, the network has been applied to a planar three-link revolute manipulator (see Section 2.7). The latter experiment clearly shows that the technique can be extended to manipulators with more than two links (see also Section 2.5). Section 2.4.3 demonstrates that the technique can be also applied to manipulators with *prismatic* joints[3], and to manipulators with a combination of prismatic and revolute joints (see also [Althoefer95b]).

2.2 The Configuration Space in Context

Obviously, each configuration of a physical manipulator with n links can be represented by a single point, or particle, in an n-dimensional C-space; this means that a robot of the size of a particle moves among transformed C-obstacles. This change of representation makes possible the use of path planning strategies which mimic physical events like the distribution of an electrical field in a charge-free space or a current flow through a conductive medium (Chapter 3).

[3] Prismatic joints constrain the links to a translational motion.

The concept of shrinking a complex robot to a point in a suitable space has been originally proposed by Udupa (see [Latombe91 and references therein]). Lozano-Pérez and Wesley further developed this idea and established the name “configuration space” [Lozano79]. The work of Lozano-Pérez *et al.* is often referred to as the first contribution to exact motion planning [Latombe91]. Based on the C-space approach, Lozano-Pérez developed path planning algorithms for robots among polygonal and polyhedral obstacles, and, furthermore, introduced the principles of approximate cell decomposition [Lozano83, Lozano87]. Since the introduction of the C-space approach many researchers have contributed to this area. For an overview see [Latombe91].

Lozano-Pérez distinguishes between two subproblems in his path planning approaches: (1) the *FindSpace* problem and (2) the *FindPath* problem [Lozano87]. The former is about generating collision-free or safe configurations, while the latter is concerned with finding a series of safe configurations which represent a collision-free path for the manipulator. C-space is a valuable tool to do path planning [Newman91, Latombe91, Lozano87]. However, its construction usually entails a great computational effort, often neglected by research. Thus, the development of fast methods to transform a workspace representation into its C-space counterpart is an important issue of research [Branicky90, Newman91, Althoefer95c, Latombe91].

The construction of the C-space is distinct from path planning. However, the transformation of the workspace into C-space can be used as a pre-processing step in many motion- or path-planning strategies (for example see Chapter 3 and [Althoefer95e, Bugmann95, Latombe91, Lozano87, Gupta90, Gupta92, Ralli96]). Not every path planning strategy calls for an explicit calculation of the configuration space. Collision detection might be evaluated along paths found in a non-complete C-space representation (see for example Chapter 4 and [Khatib86, Barraquand91]). The technique suggested in this chapter is most applicable to path planning strategies which require a precise and explicit representation of the C-space. Obstacle-free regions in this C-space can be then searched for a path.

A common approach in constructing the configuration space is to use a simplified workspace representation as a base for the transformation [Gupta92, Lozano87, Latombe91 and references therein]. Usually, obstacles and manipulator are replaced by simple polygonal objects which enclose the original. Algebraic methods are employed to determine the borderline between C-free space and C-obstacle space. Due to the crude representation of manipulator and obstacles, this approach might produce an incomplete C-space which, for example, does not contain a possible path which leads through a narrow gap between two obstacles.

The C-space transformation suggested here is based on numerical calculations and makes use of the correct geometry of the physical manipulator. It produces discrete C-space patterns for obstacle points at varying locations. These C-space patterns have been acquired in a pre-processing step where the manipulator is moved in such a way around a “point-sized” obstacle, a circular rod, that there is at all times at least one contact point between the rod and the manipulator. While tracing the rod, the manipulators internal joint sensors are constantly read and the acquired values are stored by the connected computer system. The result is a list whose entries denote all those joint angles which represent an almost-collision between the manipulator and the rod, or in other words, the perimeter of the C-space pattern. Thus, the recorded C-space patterns accurately reflect the shape of the particular manipulator. Since each C-space pattern corresponds to an obstacle point in workspace, the quality of the constructed C-space mainly depends on the resolution with which the workspace obstacles are discretised. The accuracy of the transformation also depends on the resolution of the discrete C-space patterns.[4] The proposed technique allows manoeuvring also in areas of the workspace where obstacles are lying close to each other.

Although to a certain extent similar to the one described by [Newman91, Branicky90], the technique suggested in this chapter uses a different method to store C-space patterns. Newman *et al*. report that in one experiment the boundaries of the computed C-space patterns were approximated by polygon boundaries and stored as a linked list of vertex co-ordinates; in another experiment the discretised C-space patterns were stored in a bitmap-like fashion using standard image compression techniques to reduce the memory requirements [Newman91]. The C-space patterns produced by the transformation technique proposed here is a list of vectors. For a two-link manipulator such as the MA 2000 (Figure 2.1-1), a vector pair denotes the centre line of the C-space pattern, while a third vector describes the width of the C-space pattern which reflects the width of the manipulator links. This storage method results in a very compact representation with little memory requirements. A further memory reduction is achieved by employing a neural network to “store” the obstacle-point-to-C-space-pattern relationship. The network is able to interpolate for C-space patterns which are not part of the training set.

Since the shape of a C-space pattern only depends on the distance of an obstacle point but not on its absolute location in workspace, the memory requirements for the look-up table are extremely low compared to an approach which is based on a look-up table which contains the C-space obstacle for every point in workspace [Chen92, Graf88].

Newman *et al*. report that their technique can be only applied to manipulators with revolute joints [Newman91]. This chapter will show that the workspace to C-space transformation technique can be also applied to manipulators with prismatic links and those with a combination of revolute and prismatic links. Moreover, in contrast to the approach of Newman *et al*., the technique described in this chapter can be extended to planar manipulators with *n* links.

This chapter concentrates on the workspace to C-space transformation for planar manipulators. However, the technique can be expanded and applied to certain types of three-dimensional manipulators (for example Puma robots, SCARA robots etc.). These manipulators have joints with parallel axes and a further joint whose axis is perpendicular to the others. The expanded technique has been described by Newman *et al*. [Newman91, Branicky90]. An example which shows the functionality of this approach is given in Chapter 3, Section 3.6.

Since the transformation technique presented here transforms individual obstacle points, it is especially suitable for workspace scenarios with a small number of obstacle points in it. Changes in the workspace representation caused by a small number of moving obstacles in an otherwise static environment can be dealt with in a particularly fast manner.

2.3 The Configuration Space of a Robotic Manipulator

A robotic manipulator ***A*** can be considered to be composed of *n* rigid links, with joint connections between adjacent links. The algorithms presented in this chapter are applicable to types of manipulators which have branching sections. However, for the sake of easier

[4] The quality of the subsequent planning process also depends on the resolution of the discrete map into which the generated C-space patterns are loaded (see Chapter 3).

presentation, any manipulator type investigated here has a non-branching structure where l_i ($i\in[1,2,...,n\text{-}1]$) is only connected to one following link l_j ($j\in[2,3,...,n]$) via a joint [Latombe91]. The first link, link l_1, is connected via a joint to the manipulator base which is fixed to the workspace. The free tip of the last link, l_n, is called the *end effector*. The number of joints as well as the number of links of such a manipulator $\boldsymbol{A}$ is n.

Since revolute and prismatic joints are the most commonly used types of joints in modern manipulators, the investigation in this chapter will focus on those joint types. Other joint types can be treated similarly, as reported in [Latombe91].

An element of a C-space $\boldsymbol{C}$ is denoted by $\mathbf{q}$ and represents a configuration which uniquely determines the position and orientation of each manipulator link. More precisely, a configuration $\mathbf{q}$ consists of a list of n parameters, $(q_1,\ldots,q_n)$, where each parameter q_i ($i\in[1,2,...,n]$) determines the configuration of link l_i relative to its parent link $l_{i\text{-}1}$ (see [Latombe91]). For all manipulators studied here, link l_1 is connected via a joint to the origin of the workspace.

In view of the manipulator which is used in the real-world experiments described in Section 2.8, each configuration q_i for any manipulator discussed here varies in a bounded interval $I_i = [q_i^-, q_i^+]$. Configurations q_i^- and q_i^+ denote a lower and an upper mechanical joint stop, respectively. That is, no two links l_i and l_j are supposed to collide with each other, thus, only collisions between links and obstacles are investigated.

The manipulator's workspace can be divided into two subsets. The first subset is the *obstacle workspace* and is denoted by W_B. It describes the space which is occupied by obstacles. The second subset which is the complement of the first one is denoted as *free space*. It describes the area which is free of obstacles. The obstacles in the manipulator's workspace are assumed to be rigid and are denoted by $B_1,\ldots,B_r$. Every point of B_k ($k\in[1,2,...,r]$) has a fixed position relative to the workspace origin.

2.4 The Mapping of Obstacle Points into their C-space Counterpart

2.4.1 The Single Point Mapping

The transformation technique suggested here maps single obstacle points which are situated in workspace into a corresponding C-space obstacle. This section describes the properties of the single point mapping.

Any obstacle in a discretised workspace is considered to be composed of a union of elementary obstacle units, so-called *primitives*. For each of these primitives a C-space representation can be calculated. The main purpose for using this approach is that a look-up table (Section 2.5) or associative map (Section 2.6) can be constructed which contains in each row a unique feature specifying a primitive as well as the corresponding C-space representation. Thus, any entry to the look-up table which is equal (or similar, if a neural network is used) to a stored primitive feature will cause the retrieval of the associated C-space representation which describes the C-obstacle region of the particular primitive. Different types of primitives can be used for this approach, examples include obstacle points, circular obstacles and line segments.

The investigation of the C-space representation of point obstacles is of special interest, because a point can be interpreted as the smallest unit of a discretised workspace. This unit is either a pixel in a 2-dimensional workspace or a voxel in a 3-dimensional workspace [Newman91].

From this it follows that any obstacle B_k $(k \in [1,2,...,r])$ in the manipulator's workspace can be discretised by a set of those elementary obstacle points. The discretised obstacle is denoted by B'_k. It follows that $B'_k \equiv \bigcup_{j=1}^{p_k} P_j$, where p_k is the number of obstacle points P_j contained by B'_k. The union of discretised obstacles B'_k describes the complete area of the workspace which is covered by discretised obstacles:

$$\mathbf{W}'_B \equiv \bigcup_{k=1}^{r} B'_k . \tag{2.4-1}$$

Due to the fact that the workspace is discretised, any obstacle B_k is only described up to a precision which depends on the resolution of the workspace. To assure that no collision between the manipulator and any real obstacle B_k can occur, any discretised obstacle B'_k must be constructed in such a way that it engulfs B_k completely, thus: $B'_k \supseteq B_k$. Consequently, the real free space which is the complement of $\boldsymbol{W}_B$ is either equal to the discretised free space or a subset of it.

It has been shown that the C-obstacle region of any workspace obstacle is the union of the C-space transforms of all the point obstacles which constitute the workspace obstacle [Newman91, Branicky90]. Thus, the C-space for the set of points which describes the discrete obstacle B_k is the union of C-space transforms of all point obstacles which constitute it. This can be expressed in equational form:

$$CO_A(B'_k) \equiv CO_A(\bigcup_{j=1}^{p_k} P_j) \equiv \bigcup_{j=1}^{p_k} CO_A(P_j) . \tag{2.4-2}$$

Eq. (2.4-2) shows that the C-obstacle region of manipulator $\boldsymbol{A}$ due to the obstacle B'_k is equivalent to the union of the C-space obstacles caused by each obstacle point P_j, for all $j \in [1,2,...,p_k]$. The configuration space representation of a single obstacle point P_j is denoted by $CO_{\mathrm{A}}(P_j)$ and is called here *C-space pattern*.

Equivalently, the C-space of the entire obstacle region of discretised workspace $\boldsymbol{W}'_B$ can be computed by unifying all the C-space representations of the individual obstacles. Consider the discretised workspace to be cluttered by obstacles B'_k, $k \in [1,2,...,r]$, then, as reported in [Newman91, Branicky90]:

$$CO_A(\boldsymbol{W}'_B) \equiv CO_A(\bigcup_{k=1}^{r} B'_k) \equiv \bigcup_{k=1}^{r} CO_A(B'_k) . \tag{2.4-3}$$

This means, the C-space due to $\boldsymbol{W}'_B$, which is the union of all obstacles in the discrete workspace of manipulator $\boldsymbol{A}$ is equivalent to the union of the C-obstacles of the individual obstacles. The complement of $\boldsymbol{W}'_B$ is the free space of the discretised workspace; the complement of $CO_A(\boldsymbol{W}'_B)$ is the C-space representation of the discretised free space [Newman91, Branicky90]. The fact that $CO_A(\boldsymbol{W}'_B) \supseteq CO_A(\boldsymbol{W}_B)$ assures that the robot particle which is allowed to navigate in the discretised C-free space does not collide with any C-obstacle and consequently the physical manipulator does not collide with any obstacle.

Obviously, the discretised workspace can be also described as follows:

$$\boldsymbol{W}_B' \equiv \bigcup_{k=1}^{r} B_k' \equiv \bigcup_{j=1}^{p} P_j, \tag{2.4-4}$$

where p represents the total number of point primitives in the discretised workspace $\boldsymbol{W}_B'$. It follows:

$$CO_{\boldsymbol{A}}(\boldsymbol{W}_B') \equiv CO_{\boldsymbol{A}}(\bigcup_{k=1}^{r} B_k') \equiv CO_{\boldsymbol{A}}(\bigcup_{j=1}^{p} P_j) \equiv \bigcup_{j=1}^{p} CO_{\boldsymbol{A}}(P_j). \tag{2.4-5}$$

The transformation of obstacles into C-space obstacles can be accelerated, if only those pixels (or voxels) which represent the perimeter (or surface) of the workspace obstacles are transformed. It has been shown that the C-space patterns that correspond to those pixels or voxels which surround an obstacle enclose also the corresponding C-space obstacle [Newman91]. In a 2-D application, the obstacles' perimeters can be extracted from camera images using edge-detection and thresholding techniques, as proposed in Section 2.8.3.1 (see also [Jaitly96b]).

Since the main concern of this section is to describe how to transform a particular workspace setting or scenario, obstacles in the discrete workspace are considered to be an agglomeration of obstacle primitives, each of which are transformed into the corresponding C-space pattern individually. The notion of an obstacle B_k is not so important for the further discussion, though it might have been in a pre-processing step in the image acquisition system (Section 2.8.3.1 and [Jaitly96b]). Carrying out repeatedly the described single obstacle point mapping, the complete C-space representation of any workspace scenario can be achieved.

2.4.2 The C-space of a 2-Link Revolute Arm

The basic kinematic performance of a physical manipulator in relation to obstacles located in its range can be investigated by analysing the kinematics of a "stick-like" manipulator. This kind of manipulator, as understood in this thesis, is a simplified robot arm whose links have zero width. Equivalently, the joints of this skeleton arm have zero expansion. Section 2.8 shows how these fundamental considerations can be also applied to real manipulators.

This section describes the calculation of the C-space representation for a two-link manipulator with parallel, revolute joints. Many industrial manipulators have this link-joint-link structure as part of the whole system. SCARA designs match this structure (Figure 2.1-1 (right)). Industrial manipulator of this type are: Adept manipulators, Panasonic H series, IBM 7535, GMF A-510/A-600, and RedRob [Newman91, Risse95]. The same structure can be found in the class of widespread robot types which are based on a design with three major revolute joints. This type of robot has two or three joint axes which are parallel to each other and perpendicular to the joint axis at the robot's base. Examples of this class include: all Unimation Puma manipulators (Figure 2.1-1 (middle)), all GMF S and P series manipulators, all Asea manipulators, Motoman L-series robots, most Cincinnati Millicron manipulators, and most General Electric manipulators [Newman91, Meyer88, Latombe91]. The work described here is applied to the experimental manipulator MA 2000 which has three major revolute joints (see Figure 2.1-1 (left), Section 2.8 and Appendix A-1).

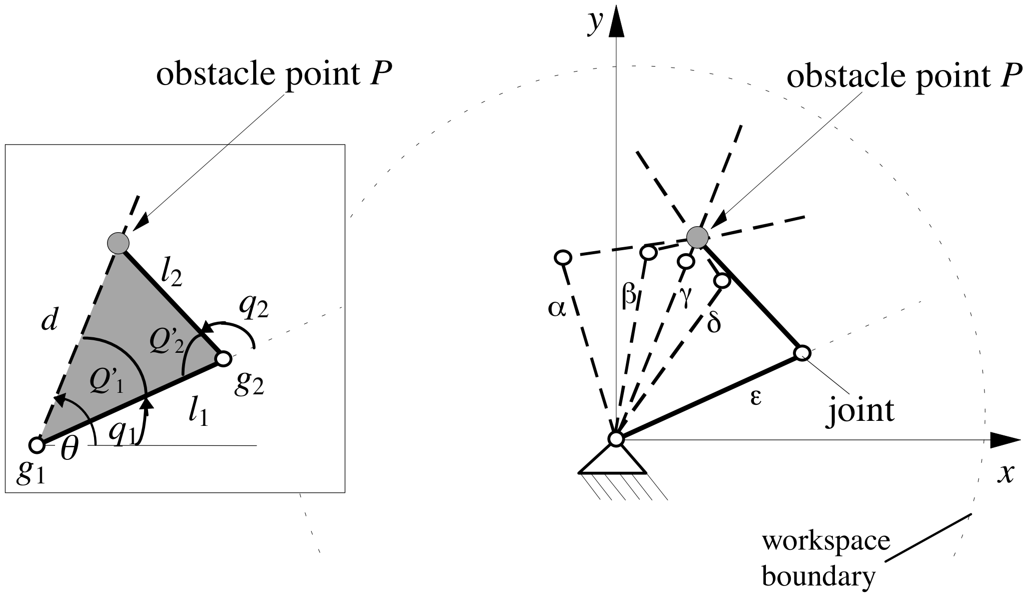


Figure 2.4-1: The two-link manipulator in its workspace. The manipulator is colliding with obstacle point P whose distance to the base is d and whose angular position is θ. Postures "α" to "ε" transform into separate configurations in C-space as shown in Figure 2.4-2. The inset on the left hand side of the figure highlights the angular variables which are used in the equations of this section. The variables q_1 and q_2 describe the configuration of the two-link arm. Variable q_1 represents the angle between link l_1 and the x-axis, while variable q_2 represents the angle between link l_1 and link l_2. The variables Q'_1 and Q'_2 are two intermediate variables which are used to derive q_1 and q_2 (see Eqs. (2.4-6) to (2.4-9)).

The two-link revolute manipulator under study in this section is depicted in Figure 2.4-1. Its first link l_1 is connected via joint g_1 to the manipulator's base which is by itself fixed to the workspace. The other end of link l_1 is connected to the succeeding link l_2 via joint g_2. In this section the mapping of a single obstacle point into the corresponding C-space pattern is investigated. This obstacle point is denoted P and is one of those pixels which describe the workspace obstacles (Section 2.4.1). Its distance to the base is denoted d.

A commonly chosen configuration space of a revolute manipulator is the joint space with parameters, q_1 and q_2, representing the angular displacements of the joints g_1 and g_2. This C-space can be depicted in a two-dimensional chart (see Figure 2.4-2). The parameters of this chart are for example chosen to be $(q_1,q_2)\in[-\pi,\pi)\times[-\pi,\pi)$ using modulo 2π arithmetic if an unbounded motion of each joint has to be taken into account. Without restrictions, the chart can be also used for manipulators whose joint parameters q_i are limited to an interval $I_i\in[-\pi,\pi]$ due to mechanical joint stops, as done in these experiments here.

Two further variables, Q_1 and Q_2, are introduced to describe a so-called *centralised C-space* which contains the unshifted C-space pattern (Figure 2.4-2). The parameters of this C-space are independent of the angular position of obstacle point P in workspace. To get the configuration space of an obstacle point in workspace, angle θ is added to the C-space pattern in the centralised space, and, hence, $(q_1,q_2)=(Q_1,Q_2)+(\theta,0)$ (see inset of Figure 2.4-1 and Figure 2.4-2). While pre-computed values of Q_1 and Q_2 are stored in a look-up table or neural network for fast access, the addition of θ is done in a separate step (Section 2.5).

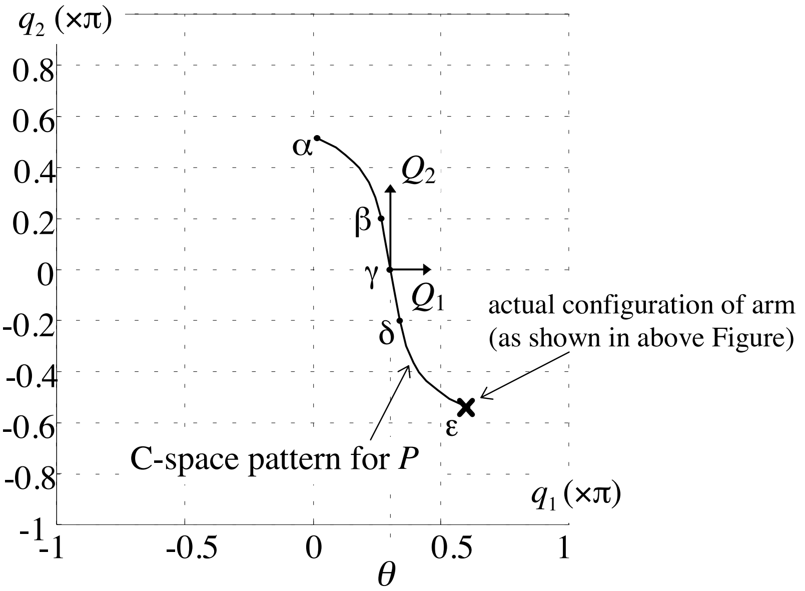


Figure 2.4-2: The configuration space which corresponds to the two-link arm in its workspace shown in Figure 2.4-1. The collision with obstacle point P in workspace transform into an s-shaped C-space pattern. The configurations "α" to "ε" correspond to the different arm positions in Figure 2.4-1. The space spanned by the co-ordinates (Q_1,Q_2) is called the centralised C-space and contains the C-space pattern. To achieve the final position of the C-space pattern in configuration space, the pattern has to be shifted by θ.

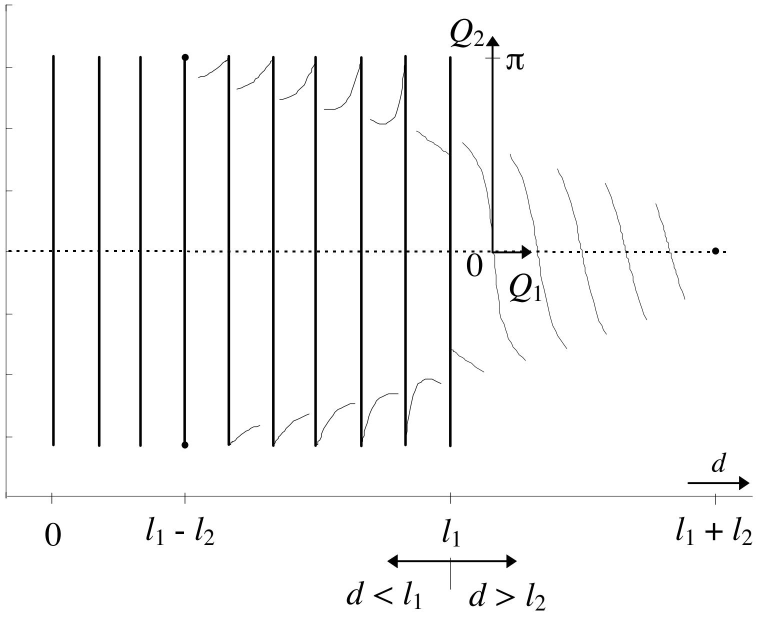


Figure 2.4-3: This figure shows C-space patterns at increments of d for a two-link manipulator like shown in Figure 2.4-1. Straight bold lines depict collisions between obstacle point and link l_1. Curved lines are patterns representing a collision caused by link l_2. The C-space patterns shrink to points at $d = l_1 + l_2$ and $d = l_1 - l_2$, respectively. Note that the d-axis indicates the distance between the base of the manipulator and an obstacle point P.

As one can see in Figure 2.4-3, the C-space caused by the collision between an obstacle point in workspace and any part of the stick-like manipulator is a curve or a combination of a curve and a straight line. The calculation of the curved part of a C-space pattern can be derived from the inverse kinematic equations for robotic manipulators (Eqs. (2.4-10) and (2.4-11)) which describe in their basic form the collisions of the manipulator's end effector with an obstacle point. Thus, those equations determine the value of the joint parameters q_1 and q_2 for an end effector position given by Cartesian co-ordinates x and y. The solutions of these equations correspond to the two end points of the C-space patterns. The following paragraphs will explain how one can derive the complete C-space pattern.

The inverse kinematic equations (Eqs. (2.4-10) and (2.4-11)) are based on the law of cosines for triangles (see also inset of Figure 2.4-1):

$$l_2^2 = d^2 + l_1^2 - 2dl_1 \cos(Q_1') \qquad (2.4\text{-}6)$$

and

$$d^2 = l_1^2 + l_2^2 - 2l_1 l_2 \cos(Q_2'), \qquad (2.4\text{-}7)$$

where $d = \sqrt{d_x^2 + d_y^2}$ describes the distance between manipulator base and obstacle point P and where variables Q_1' and Q_2' denote the two angles in the triangle formed by the two links l_1 and l_2 (see inset of Figure 2.4-1). The desired joint parameters q_1 and q_2 have the following relationship with the angles Q_1' and Q_2':

$$q_1 = \theta + elbow \cdot Q_1', \qquad (2.4\text{-}8)$$

and

$$q_2 = elbow \cdot (Q_2' - \pi), \qquad (2.4\text{-}9)$$

where *elbow* is equal to +1 and -1 corresponding to the elbow link (l_2) being up (left-handed) and down (right-handed), respectively [Maciejewski93, Fox92]. The angular parameters q_1 and q_2 can be derived from Eqs. (2.4-6) and (2.4-7), as follows:

$$q_1 = \theta + elbow \cdot \cos^{-1} \frac{d^2 + l_1^2 - l_2^2}{2l_1 d} \qquad (2.4\text{-}10)$$

and

$$q_2 = elbow \cdot (\cos^{-1} \frac{l_1^2 + l_2^2 - d^2}{2l_1 l_2} - \pi), \qquad (2.4\text{-}11)$$

where $\theta = arctan2(y,x)$. The function *arctan2* is the four-quadrant inverse tangent of the real parts of the elements of y and x. The resulting θ is in the range: $\theta \in [-\pi, +\pi]$. The function *arctan2* is derived from the inverse trigonometric function *arctan*, as follows:

$$arctan2(y,x) = \begin{cases} arctan(y/x) & x > 0 \\ \pi/2 & x = 0 \wedge y > 0 \\ 0 & x = 0 \wedge y = 0 \\ -\pi/2 & x = 0 \wedge y < 0 \\ arctan(y/x) - \pi & x < 0 \wedge y \leq 0 \\ arctan(y/x) + \pi & x < 0 \wedge y > 0 \end{cases} . \qquad (2.4\text{-}12)$$

The two Eqs. (2.4-10) and (2.4-11) allow the calculation of the configurations for which the manipulator's end effector touches obstacle point *P*. However, the C-space pattern describes all possible collisions between any part of the manipulator and the obstacle point. To compute the C-space pattern, two main cases can be distinguished: firstly, collisions with link l_1, secondly, collisions with link l_2. Firstly, whenever a collision between link l_1 and an obstacle point occurs the corresponding C-space pattern is a vertical strip which includes all possible values of q_2:

$$q_1 = \theta ,$$

$$q_2 = [q_2^-, q_2^+].$$

Note, a collision between link l_1 and an obstacle results in a forbidden area regardless of the configuration of link l_2 (see Figure 2.4-3). Secondly, collisions between link l_2 and obstacle point *P* can be derived by considering link l_2 to be a variable [Maciejewski93]. Thus, there are not only two solutions for Eqs. (2.4-10) and (2.4-11), but a one-dimensional infinity of solutions prarameterized by the variable, l_2' [Maciejewski93]. Varying l_2' in the range $[l_1 - l_2, l_1 + l_2]$ the C-space pattern for any obstacle point in the distance *d* can be calculated (see Figure 2.4-3). This process corresponds to a motion of the manipulator while link l_2 is sliding through obstacle point *P* in distance *d*, with $d \in [l_1 - l_2, l_1 + l_2]$ (see also Figure 2.4-1). Start and end configuration of this motion are described by Eqs. (2.4-10) and (2.4-11) with link l_2 being fixed to its original length.

For any revolute manipulator, the shape of a C-space pattern depends only on distance *d* between the manipulator base and the obstacle point [Newman91]. The position of a C-space pattern in the configuration space depends on θ which causes a shift along q_1 (see Figure 2.4-2). The centre of any pattern lies at $Q_1 = 0$, $Q_2 = 0$ in the centralised C-space. This centre configuration describes the arm being in a fully stretched pose pointing horizontally in the direction of the positive *x*-axis. Figure 2.4-1 and Figure 2.4-2 show how different poses of a manipulator, which is in contact with obstacle point *P*, are transformed into a C-space pattern. The extreme points in the C-space pattern "α" and "ε" correspond to the two configurations where only the end effector of the arm is in contact with the point *P*. For all intermediate configurations link l_2 is "sliding through" point *P*. C-space patterns stored in a look-up table can be employed to quickly construct the configuration space for a given workspace scenario (see Section 2.5).

2.4.3 The C-space of a 2-Link Arm with Prismatic and Revolute Joints

Most research work concerning the workspace to C-space transformation deals with manipulator types whose links are connected by revolute joints (for example see [Latombe91, Lozano87, Gupta92]. The treatment of manipulator types with prismatic joints or a combination of revolute and prismatic joints is not discussed widely in the robotics research community. Newman *et al*. report that a workspace to C-space transformation is only possible for manipulators with revolute joints [Newman91].

This section here shows an extension of the technique calculating the C-space patterns for revolute-revolute manipulators (Section 2.4.2). The extension permits the calculation for manipulator types with a combination of prismatic-revolute joints, revolute-prismatic joints and prismatic-prismatic joints (see also [Althoefer95b].

As in the above considerations (Section 2.4.2), each of the investigated manipulator types has two stick-like links l_1 and l_2. The first link l_1 is connected via joint g_1 to the base of the manipulator. The other end of link l_1 is connected to the succeeding link l_2 via joint g_2. The axes of the joints are parallel. Four different manipulator types with the following joint combinations are discussed:

R-R: a manipulator with two revolute joints,

P-R: a manipulator with a prismatic joint at the base and a revolute joint between links l_1 and l_2,

R-P: a manipulator with a revolute joint at the base and a prismatic joint between links l_1 and l_2,

P-P: a manipulator with two prismatic joints.

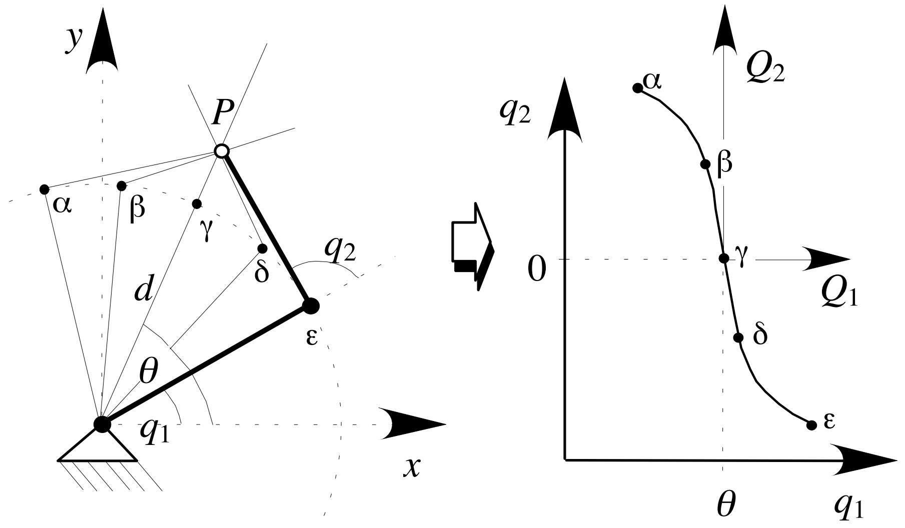
As shown in Section 2.4.2, the co-ordinates of the C-space of the **revolute-revolute** manipulator were the two angular parameters q_1 and q_2. For this manipulator type the length of the links are the constant values l_1 and l_2.

The **prismatic-revolute** type has a prismatic joint at the base, thus, the first co-ordinate is the now variable l_1. The second co-ordinate is q_2 representing the joint parameter of the revolute joint. This manipulator type has a fixed angle q_1 at the base and a constant link l_2. Since angle q_1 is fixed, the C-space of this manipulator type is independent of q_1, and the manipulator could be also depicted in a workspace whose x-axis coincides with link l_1. For this case the inverse kinematic equations can be simplified as shown in Table 2.4-2 (P-R). The shape of the C-space patterns for this manipulator depends only on distance d_y between link l_1 and obstacle point P, while d_x describes a shift of the C-space pattern in the configuration space (see Figure 2.4-4 (P-R)).

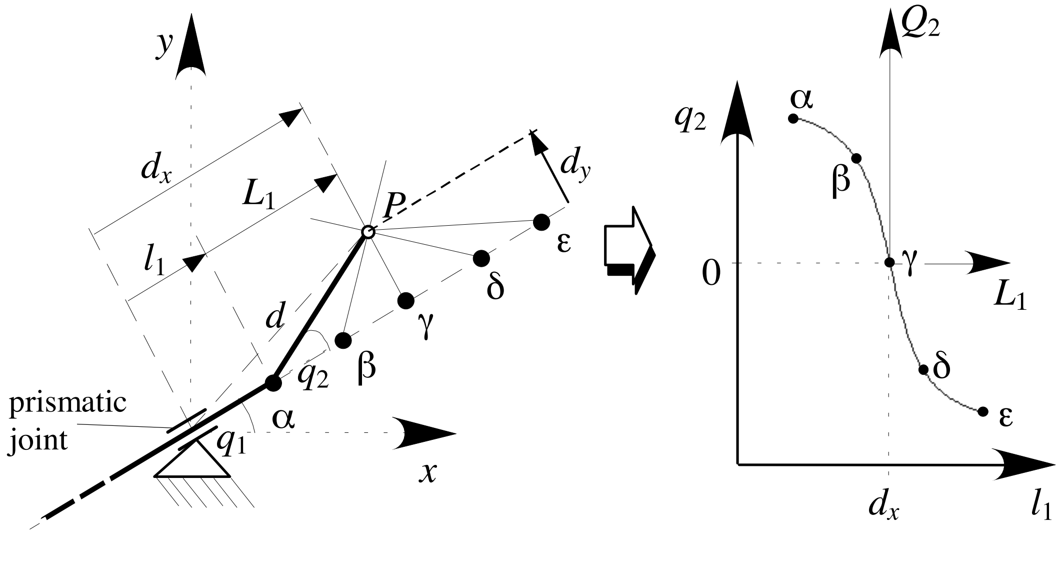
The **revolute-prismatic** type has a C-space with the co-ordinates q_1 and l_2, while angle q_2 and link l_1 are fixed. The C-space patterns representing collisions between link l_2 of this manipulator type and an obstacle point are interconnected straight lines shifted by angle θ in C-space (Figure 2.4-4 (R-P)). Although the angle of q_2 in Figure 2.1-1 (R-P) is set to $\pi/2$, other values for q_2 would also produce straight line patterns.

The **prismatic-prismatic** type has two prismatic joints, thus, its C-space is spanned by the co-ordinates l_1 and l_2, and angles q_1 and q_2 are fixed. As was the case with the prismatic-revolute manipulator, the prismatic-prismatic manipulator can be depicted in a workspace whose x-axis coincides with the link l_1 resulting in a simplification of the kinematic equations (Table 2.4-2 (P-P)). The C-space patterns are straight lines whose shape depends only on d_y and the location of the patterns in configuration space depends on d_x. The two fixed manipulator angles, q_1 and q_2, depicted in Figure 2.4-4 (P-P) are π/2; other angle values would also result in straight line C-space patterns.

The co-ordinates of the C-space for each manipulator and the constant parameters are summarised in Table 2.4-1.

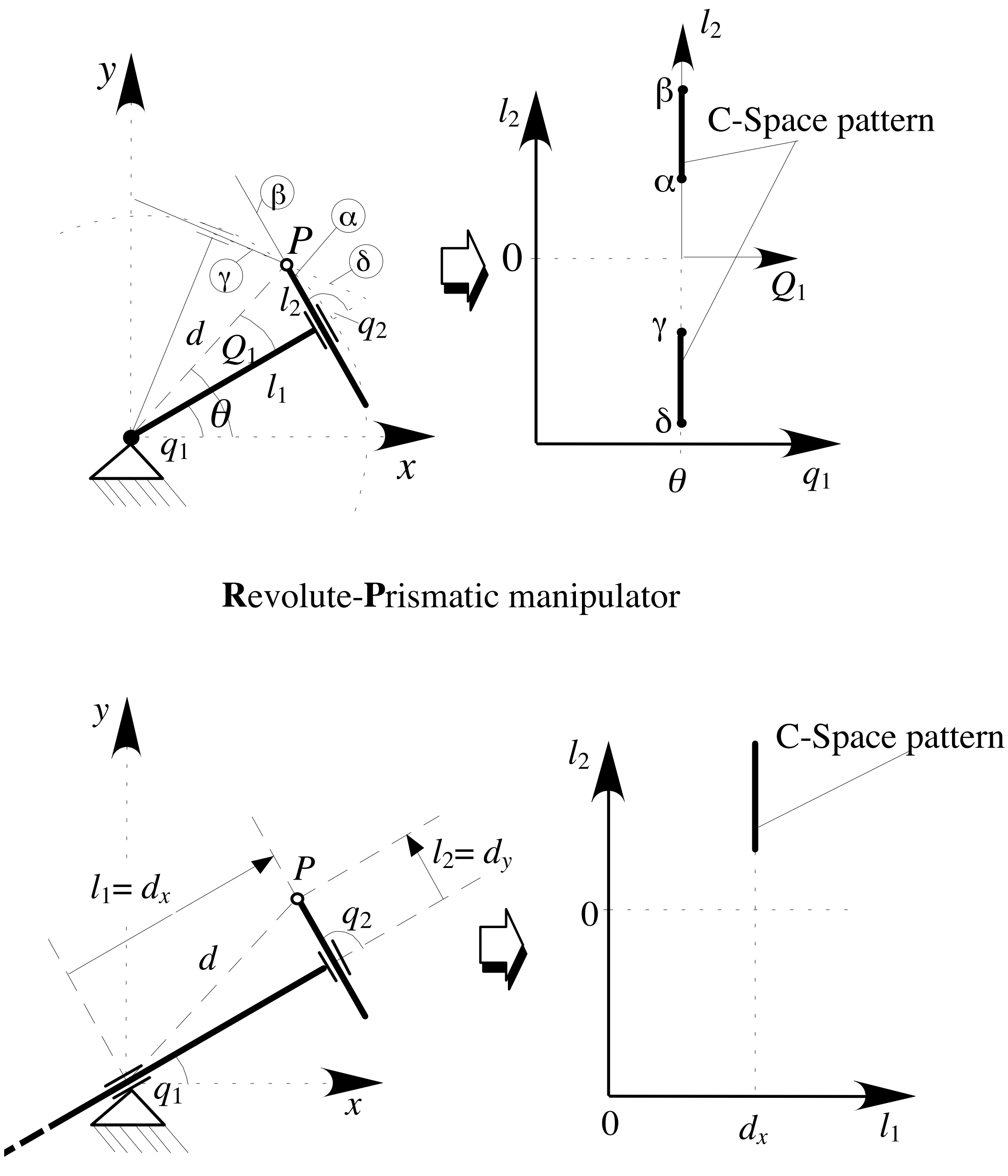


Revolute-**R**evolute manipulator



Prismatic-**R**evolute manipulator

(See caption next page)



Revolute-**P**rismatic manipulator

Prismatic-**P**rismatic manipulator

Figure 2.4-4: Depiction of the four possible manipulator types in workspace and C-space. The revolute-prismatic type as well as the prismatic-prismatic type produce C-space patterns which are straight lines vertical to the abscissa. Point *P* is an obstacle point at distance *d* from the base. The set of all possible collisions with point *P* transforms into a C-space pattern. The C-space parameters depend on the joint variables of the manipulator type (refer to Table 2.4-1).

	R-R	P-R	R-P	P-P
C-space co-ordinates	q_1, q_2	l_1, q_2	q_1, l_2	l_1, l_2
constants	l_1, l_2	q_1, l_2	l_1, q_2	q_1, q_2

Table 2.4-1: C-space co-ordinates and constants of the four manipulator types. All constants are positive values (see also Figure 2.4-4).

R-R	$q_1 = \theta \pm \cos^{-1}\frac{d^2 + l_1^2 - l_2^2}{2l_1 d}$ $q_2 = \pm(\cos^{-1}\frac{l_1^2 + l_2^2 - d^2}{2l_1 l_2} - \pi)$
P-R *)	$l_1 = d \cdot \cos(Q_1) \pm \sqrt{l_2^2 - d^2 \sin^2(Q_1)} = d_x - l_2 \cdot \cos(q_2)$ $q_2 = \pm(\cos^{-1}\frac{l_1^2 + l_2^2 - d^2}{2l_1 l_2} - \pi) = \sin^{-1}\frac{d_y}{l_2}$
R-P	$q_1 = \theta \pm \cos^{-1}\frac{d^2 + l_1^2 - l_2^2}{2l_1 d}$ $l_2 = l_1 \cdot \cos(Q_2) \pm \sqrt{d^2 - l_1^2 \sin^2(Q_2)}$
P-P *)	$l_1 = d \cdot \cos(Q_1) \pm \sqrt{l_2^2 - d^2 \sin^2(Q_1)} = d_x$ $l_2 = l_1 \cdot \cos(Q_2) \pm \sqrt{d^2 - l_1^2 \sin^2(Q_2)} = d_y$

Table 2.4-2: The inverse kinematics equations for the four different manipulator types.
*) Without loss of generality, the angle q_1 which is a constant for the **P-R**-type and the **P-P**-type is set to zero.

Eqs. (2.4-6) and (2.4-7) (see Section 2.4.2) can be used to describe the geometrical properties of all four manipulator types. Rearranging those equations, one obtains the inverse kinematics equations for the four arm types (see Table 2.4-2).

As one can see from Figure 2.4-4, the prismatic-revolute manipulator produces C-space patterns which are shaped similarly to those of the revolute-revolute manipulator. Thus, a look-up table which stores C-space patterns for obstacle points in varying distances is useful for this manipulator type to provide a rapid workspace to C-space transformation [Althoefer95b]. The two other manipulator type produce rather simple C-space patterns. For these types, an analytical computation of the C-space patterns might be more suitable.

2.5 The C-space of an *n*-Link Arm

This section shows that employing the 2-link transformation technique (Section 2.4.2) in a recursive manner, C-space patterns for planar manipulators with *n* revolute joints can be composed.

A non-branching robotic manipulator $\boldsymbol{A}$ with n planar links is considered here. Manipulator $\boldsymbol{A}$ is composed of n-1 elementary subarms $\boldsymbol{S}^i$, $i \in [1,n-1]$, where each subarm $\boldsymbol{S}^i$ consists of two links, K^i and L^i (see Figure 2.5-1). Those subarms have C-space patterns like the two-link arms described in Section 2.4.2. Link K^i represents link l_i of manipulator $\boldsymbol{A}$, while link L^i is a straight link which is composed of all succeeding links, $l_{i+1},\ldots,l_n$, with joint parameters $q_{i+1},\ldots,q_n$ set to 0. This means, links $l_{i+1},\ldots,l_n$ are fully stretched and resemble a straight line. The length of L^i is the sum of the length of its components, $l_{i+1},\ldots,l_n$. A subarm $\boldsymbol{S}^i$ has only two active joints, F^i and G^i. The first joint F^i connects link K^i to a virtual base. The second joint G^i connects link K^i with link L^i (see Figure 2.5-1).

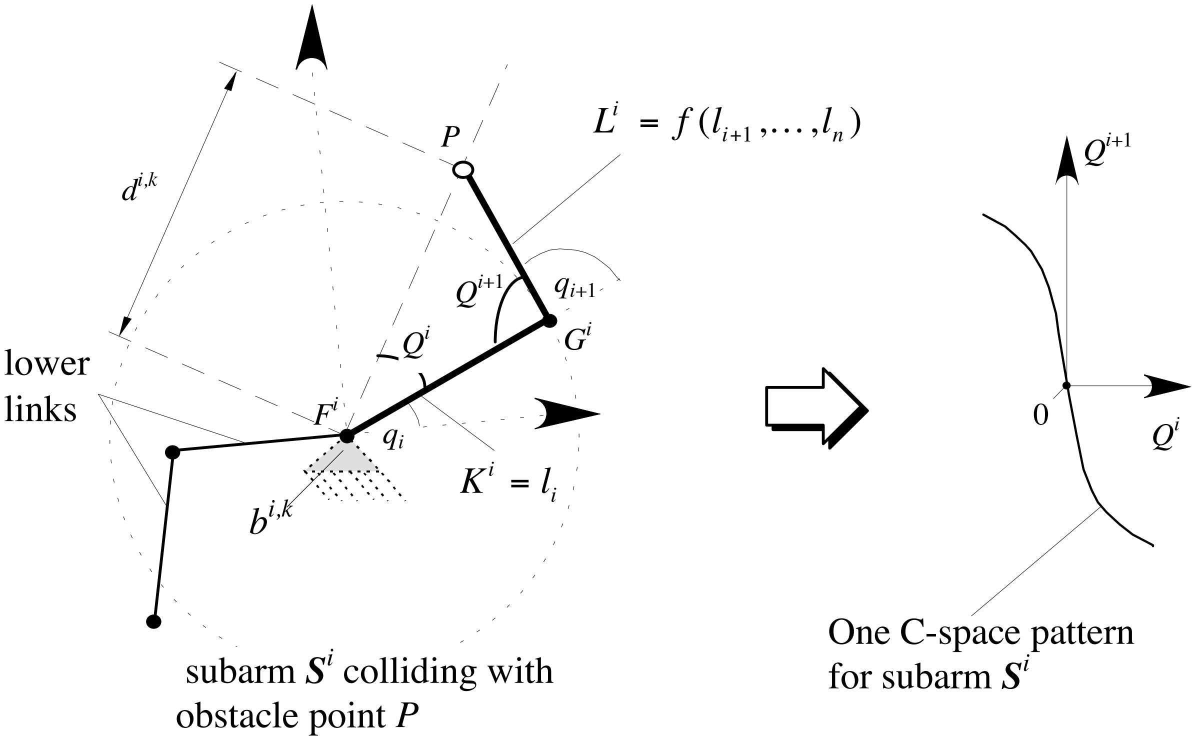

$d^{i,k}$
P
$L^i = f(l_{i+1},\ldots,l_n)$
Q^{i+1}
q_{i+1}
Q^i
G^i
lower links
F^i
q_i
$K^i = l_i$
$b^{i,k}$
Q^{i+1}
0
Q^i
subarm $\boldsymbol{S}^i$ colliding with obstacle point P
One C-space pattern for subarm $\boldsymbol{S}^i$

Figure 2.5-1: Collision between obstacle point P and one of the subarms (comprising K^i and L^i) of a redundant planar manipulator. The base $b^{i,k}$ is a virtual base as described in the text.

For a better understanding of the concept “subarm”, this section describes how the subarms are constructed from the first link of manipulator $\boldsymbol{A}$ to its last link. The first subarm $\boldsymbol{S}^1$ is composed of K^1 (which is equal to l_1) and L^1 (which incorporates in the above described manner links $l_2,\ldots,l_n$). The second subarm $\boldsymbol{S}^2$ consists of K^2 (which is equivalent to l_2) and L^2 (which embodies links $l_3,\ldots,l_n$). This process can be continued until subarm $\boldsymbol{S}^{n\text{-}1}$ is reached. This last subarm consists of $K^{n\text{-}1}$ (which is equivalent to l_{n-1}) and $L^{n\text{-}1}$ (which is equivalent to the last link, l_n). For each of those two-link subarms the C-space patterns due to point obstacles can be computed or recorded (see also [Althoefer94, Althoefer95c, Newmann91]). The technique described in the following paragraphs actually calculates C-space patterns only for collisions between links L^i and the obstacle point. Collisions between K^i and the obstacle point result in straight lines which are calculated in a separate stage.

To calculate the C-space pattern for one obstacle point with regard to an n-link arm, the following procedure which makes use of the subarm concept is invoked. The process starts with subarm $\boldsymbol{S}^1$. The collisions between $\boldsymbol{S}^1$ and obstacle point P result in a two-dimensional C-space pattern. As already stated, the shape of the C-space pattern only depends on the distance between obstacle point and the manipulator base, b^1. The pattern is comprised of discrete configurations (see Figure 2.5-3). The pattern is discretised so that it can be stored in a look-up table or neural network, as explained in a later part of this section. For each of these configurations the distance between the position of G^1 (which is the joint between link K^1 and link L^1) and obstacle point P can be calculated. Thus, the positions of G^1 become the virtual bases $b^{i,k}$ for subarm $\boldsymbol{S}^2$. For each of these virtual bases a new two-dimensional C-space pattern with respect to subarm $\boldsymbol{S}^2$ is calculated. Each of the multiple C-space patterns of subarm $\boldsymbol{S}^2$ is placed in configuration space in such a way that the pattern's centre coincides with the corresponding configuration of the previous C-space pattern (Figure 2.5-4). (The corresponding configuration is the one which is associated with the position of G^1 calculated earlier.) The C-space patterns of subarm $\boldsymbol{S}^2$ are placed perpendicular to the previous C-space pattern thereby spanning a now three-dimensional space. Informally, the C-space patterns of subarm $\boldsymbol{S}^2$ grow perpendicular out of the discrete configurations of the previous C-space pattern (Figure 2.5-4). The newly generated C-space patterns are also made up of discrete configurations. Again, for each of these configurations the subarm's joint positions can be used as virtual bases for the following subarm. The process continues in the above described manner adding with each subarm a new dimension to the configuration space. The last C-space patterns to be calculated are those of subarm S^{n-1}.

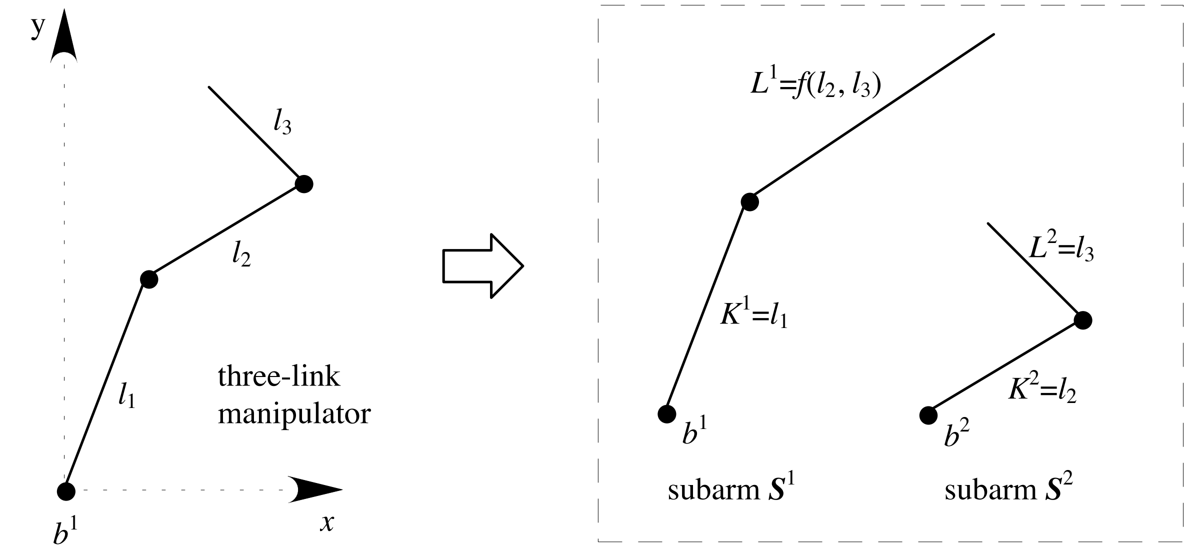


Figure 2.5-2: The construction of subarms for a three-link manipulator with stick-like links.

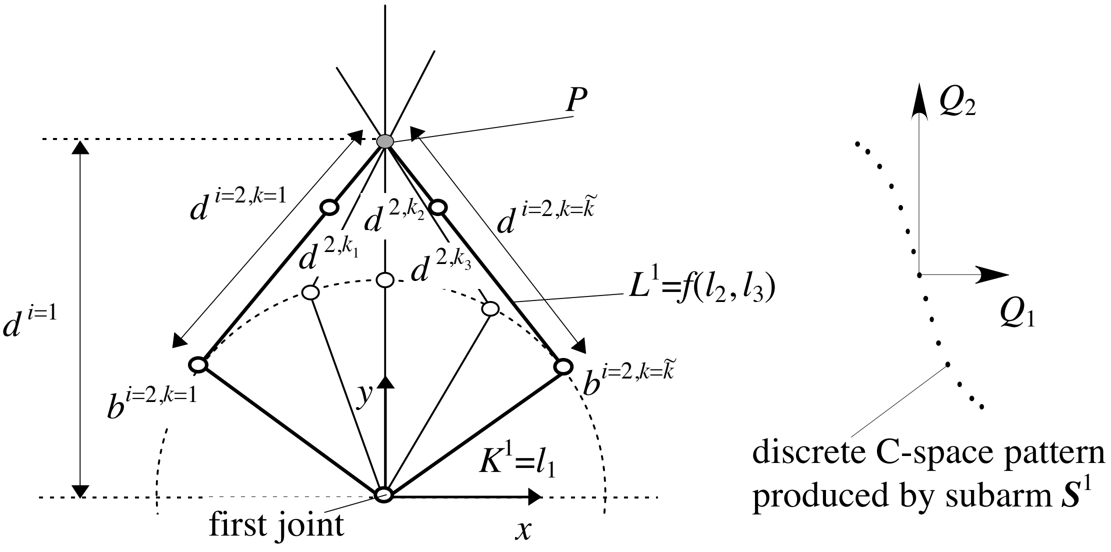


Figure 2.5-3: A three-link manipulator colliding with point *P*. The figure shows the first stage of the subarm procedure. Collisions of subarm $\boldsymbol{S}^1$ produce the discrete C-space pattern on the right. At each of the configurations of the discrete C-space pattern, a C-space pattern of subarm $\boldsymbol{S}^2$ is placed, as schematically depicted in Figure 2.5-4. The new C-space patterns are due to collisions between obstacle point *P* and subarm $\boldsymbol{S}^2$ being placed successively at bases $b^{2,k}$.

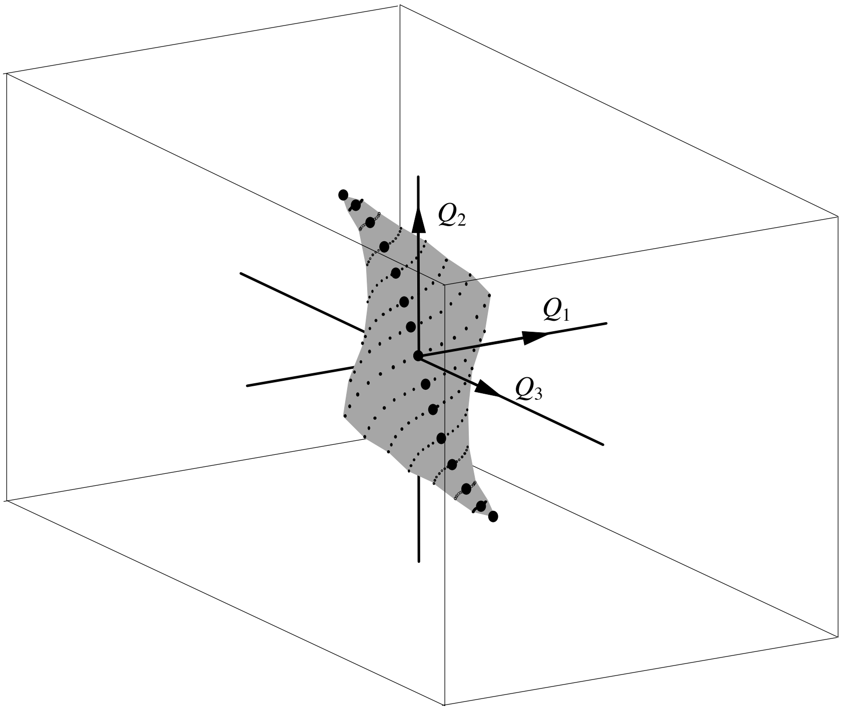


Figure 2.5-4: Schematic depiction of the C-space pattern of a three-link planar manipulator due to a collision with obstacle point *P* (see Figure 2.5-3). This C-space pattern describes a surface (shaded area) in a three-dimensional space. The "spine" of the C-space pattern (denoted by larger black points) describes the collisions between obstacle point and subarm $\boldsymbol{S}^1$ (see Figure 2.5-3). The "ribs" are the C-space patterns due to collisions of subarm $\boldsymbol{S}^2$ and obstacle point *P*. Subarm $\boldsymbol{S}^2$ is consecutively fixed to the virtual bases $b^{i,k}$ shown in Figure 2.5-3.

To avoid lengthy computations it is useful to create a list in which the C-space patterns for elementary obstacle points at increments of d are stored. The creation of this list is done in a pre-processing stage. For each subarm such a list has to be established. The list contains two columns (see Table 2.5-1). The first column contains a set of scalar parameters $d^{i,k}$ which cover the range of subarm $\boldsymbol{S}^i$ along one axis. This range is discretised by resolution $\tilde{k}$, $(k \in [1,\ldots,\tilde{k}])$, also referred to as resolution of input space (see also Section 2.6). A parameter $d^{i,k}$ represents the distance between bases $b^{i,k}$ of subarm $\boldsymbol{S}^i$ and an obstacle point P. The second column holds in each row a triple of vectors $\mathbf{Q}_1^{i,k}, \mathbf{Q}_2^{i,k}, \mathbf{d}^{i+1,k}$ which is associated with the parameter $d^{i,k}$. Hence, the list can be used as a look-up table: for any distance parameter being the entry to the list or look-up table, one can retrieve the corresponding vector triple containing a 2-dimensional C-space pattern (Q_1,Q_2) and a vector of distances whose elements are pointers to the list of the following subarm. The proposed list can be easily extended for manipulators with non-zero link widths by adding another column which reflects the widths of C-space patterns (compare to the C-space patterns in Section 2.8.1 and Appendices A-2 and A-3).

column 1	column 2		
$d^{i,k=1}$	$\mathbf{Q}_1^{i,k=1}$	$\mathbf{Q}_2^{i,k=1}$	$\mathbf{d}^{i+1,k=1}$
.	.	.	.
.	.	.	.
.	.	.	.
$d^{i,k=\tilde{k}}$	$\mathbf{Q}_1^{i,k=\tilde{k}}$	$\mathbf{Q}_2^{i,k=\tilde{k}}$	$\mathbf{d}^{i+1,k=\tilde{k}}$

Table 2.5-1: The look-up table shows the list for subarm $\boldsymbol{S}^i$. Each distance value in column 1 is associated with the vector triple in the same row. The distance vector in the very last column can be used as entry to the look-up table of the next subarm, $\boldsymbol{S}^{i+1}$.

The vectors of the vector triple $\mathbf{Q}_1^{i,k}, \mathbf{Q}_2^{i,k}, \mathbf{d}^{i+1,k}$ can be denoted as follows:

$$\mathbf{Q}_1^{i,k} = (Q_1^{i,s=1},\ldots,Q_1^{i,s=\tilde{s}}),$$

$$\mathbf{Q}_2^{i,k} = (Q_2^{i,s=1},\ldots,Q_2^{i,s=\tilde{s}}), \qquad (2.5\text{-}1)$$

$$\mathbf{d}^{i+1,k} = (d^{i+1,s=1},\ldots,d^{i+1,s=\tilde{s}}),$$

where $s \in [1,2,...,\tilde{s}]$ and $\tilde{s}$ is the resolution of the output space (see also Section 2.6.1).

The first two vectors $\mathbf{Q}_1^{i,k}, \mathbf{Q}_2^{i,k}$ represent the two-dimensional C-space pattern that is caused by the collision between link L^i of subarm $\boldsymbol{S}^i$ and the obstacle point which is at a distance $d^{i,k}$ from base $b^{i,k}$. This C-space pattern is sampled by the elements of the vectors $\mathbf{Q}_1^{i,k}, \mathbf{Q}_2^{i,k}$. In other words, each pair of angles $Q_1^{i,k}, Q_2^{i,k}$ of vectors $\mathbf{Q}_1^{i,k}, \mathbf{Q}_2^{i,k}$ describes a single subarm configuration; the set of those configurations represents the discretised C-space pattern for subarm $\boldsymbol{S}^i$ due to a collision with obstacle point P. Every configuration (or pair of joint parameters) is associated with one of the distance parameters $d^{i+1,s}$ in the third vector $\mathbf{d}^{i+1,k}$. Each parameter $d^{i+1,s}$ describes the distance between $b^{i+1,s}$ and the obstacle point, while the subarm $\boldsymbol{S}^i$ is in configuration $Q_1^{i,s}, Q_2^{i,s}$. Each parameter $d^{i+1,s}$ functions as an entry to the list of the next two-link subarm, $\boldsymbol{S}^{i+1}$. This process can be continued until the last subarm, $\boldsymbol{S}^{n-1}$, is reached, thus, the C-space pattern of an n-link planar manipulator can be constructed by accessing recursively or in parallel the look-up tables for all subarms of the manipulator [Althoefer94]. This process is depicted for a three-link manipulator in Figure 2.5-5. The presented technique is especially fast for manipulators with a low degree of freedom, but it can cope with n degrees of freedom as well. The order of complexity of this technique is O(a^n), where a is a constant representing the calculation for one manipulator link and n is the number of manipulator links.

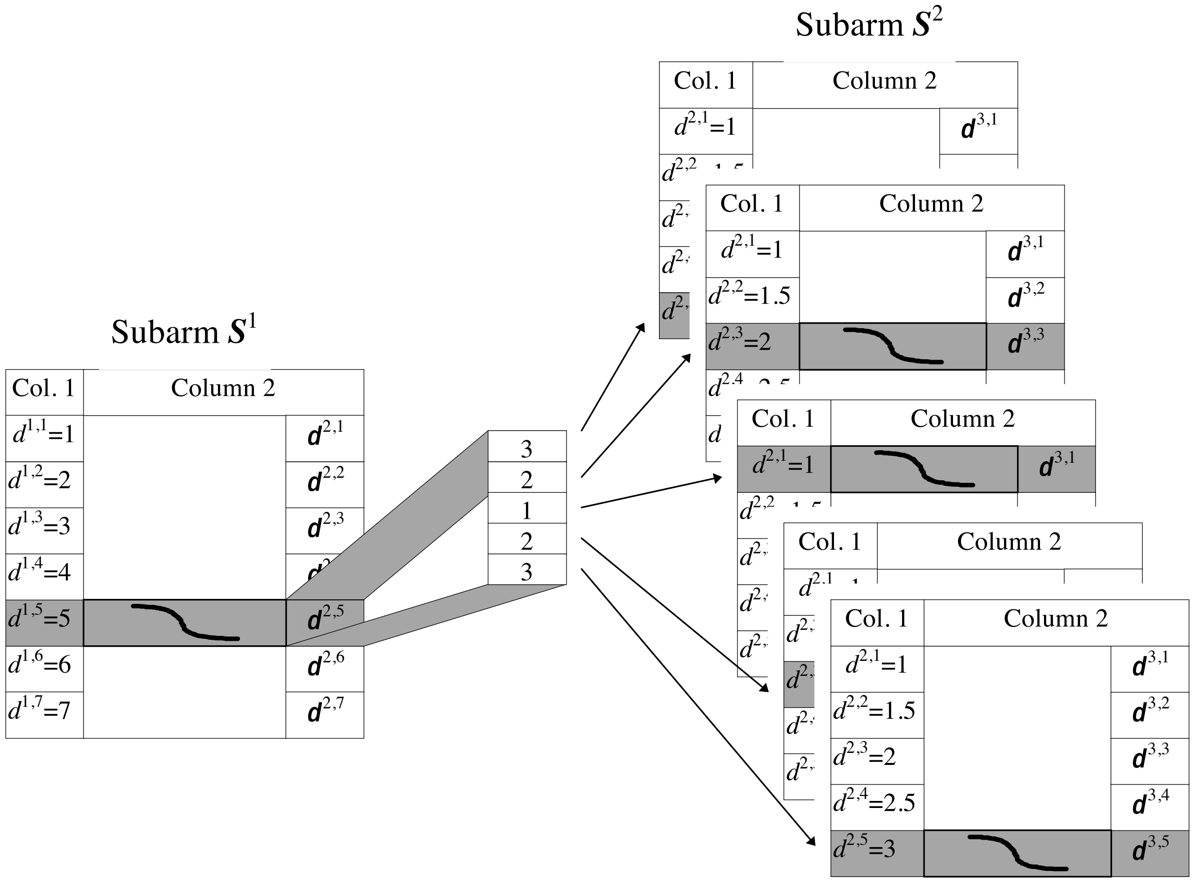


Figure 2.5-5: Depiction of the accessing of the look-up tables of a three-link manipulator. At first a distance value $d = 5$ is fed to the look-up table of subarm S^1. This triggers the retrieval of the corresponding C-space pattern and the vector $\mathbf{d}^{2,5}$ which contains distance values $d^{2,k}$ for subarm S^2. Each of these distance values triggers a C-space pattern in the table of subarm S^2. Note that there exists only one look-up table for each subarm, but the table of subarm S^2 is accessed five times in this figure. The depicted process can be carried out recursively or in a parallel manner (see Figure 2.5-6). The neural network presented in Section 2.6 can be used to "store" the data of these look-up tables.

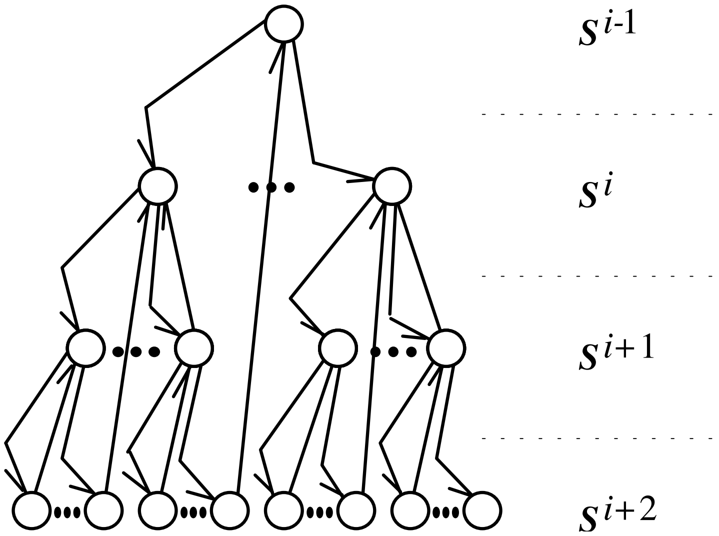


Figure 2.5-6: Tree structure representing the computation of the C-space patterns. Each node depicts the single access to a subarm look-up table. If the computation is done recursively, the nodes are accessed in the order the half-arrows indicate (from top to bottom and from left to right). In a parallel implementation the nodes on one level could be substituted by processors which share the memory.

2.5.1.1 Comparison of configuration space representations

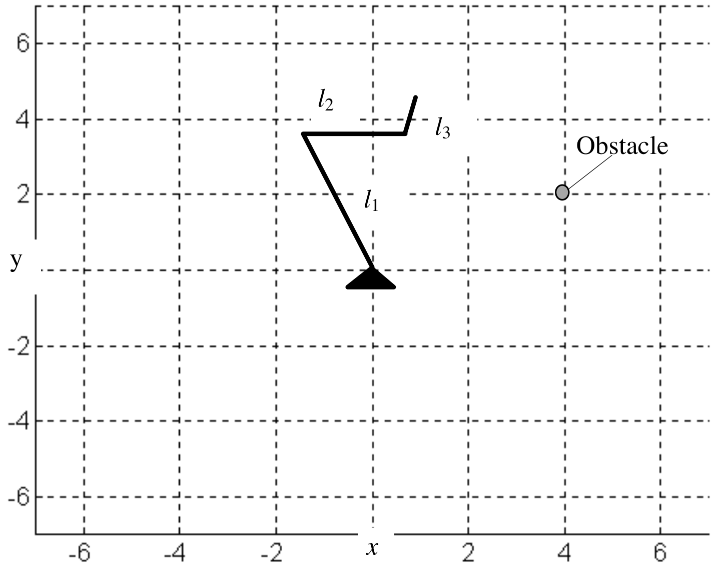


Figure 2.5-7: This figure depicts the planar workspace of a three-link revolute manipulator. One obstacle point is placed at $x_1 = 3.98$, $y_1 = 2.01$. The robot manipulator is a simulated arm with revolute joints and stick-like links. Length of the three links: $l_1 = 4$ units, $l_2 = 2$ units, $l_3 = 1$ unit.

To check whether the technique proposed in Section 2.5 produces the desired output, the following test has been carried out. The manipulator's links (Figure 2.5-7) have been moved in small angular increments covering the whole range of the joints. Every time a collision between arm and obstacle occurred, the corresponding configuration was marked in the C-space chart with a small pixel. This simple and well-known, but time consuming technique is an exhaustive workspace scan whose resolution and execution time depends on the size of the angular increments. This technique is used here to show the functioning of the technique presented in Section 2.5.

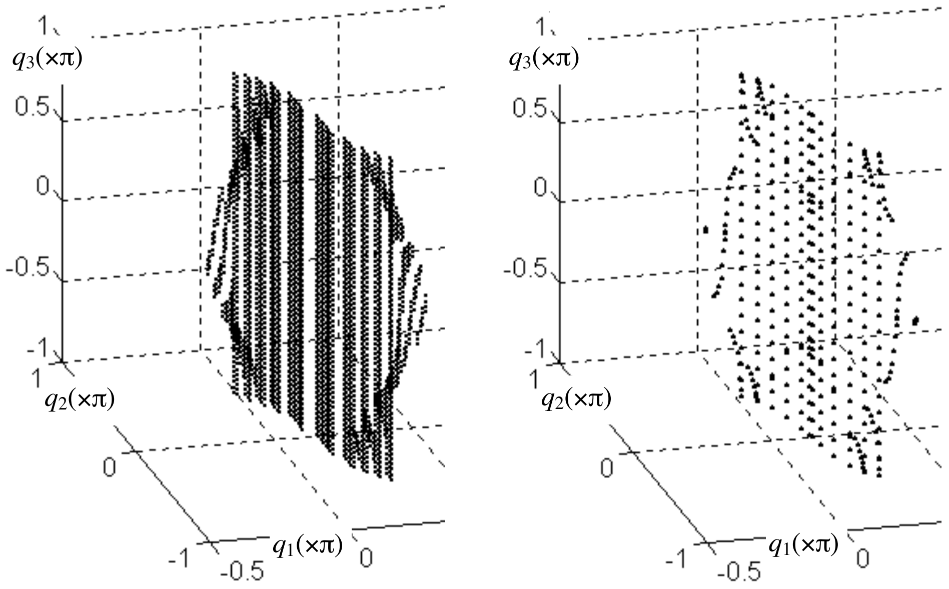


Figure 2.5-8: This figure depicts two configuration space representations of a three-link planar manipulator in its workspace (see Figure 2.5-7). The configuration space on the left is produced by rotating the manipulator's links through a sequence of configurations at angular increments. Whenever a collision with the obstacle point (placed at $x_1 = 3.98$, $y_1 = 2.01$) occurs, the corresponding configuration (q_1, q_2, q_3) is marked in C-space. The right figure shows the configuration space obtained by using the subarm technique proposed in Section 2.5. Obviously, the C-space obstacle of both figures cover a similar area; variations are due to the different methods of working of the two approaches.

Figure 2.5-8 compares the C-space representations generated by the two different techniques. The pattern on the left was generated by the test technique, and the pattern on the right shows the C-space pattern of the manipulator produced by the workspace to C-space transformation technique proposed in Section 2.5. The latter technique does not scan the workspace for the occurrence of collisions by moving the manipulator, but maps each workspace obstacle point into C-space patterns stored in look-up tables. Its resolution depends on the resolution of the stored C-space patterns, and its execution time mainly depends on the number of obstacle points in workspace. As one can see in Figure 2.5-8, the two C-space patterns cover a similar area. Variations are due to the different techniques of working of the two approaches.

2.5.1.2 Reduction in Complexity

The workspace to C-space transformation method described has one main disadvantage. It is applied to the entire n-dimensional C-space and therefore suffers from the inherent dimensionality problem. The extended approach which is outlined in this section for a planar manipulator allows the decoupling of the n-dimensional configuration space into lower dimensional ones. This idea is based on what is described as "slicing in configuration space" [Latombe91].

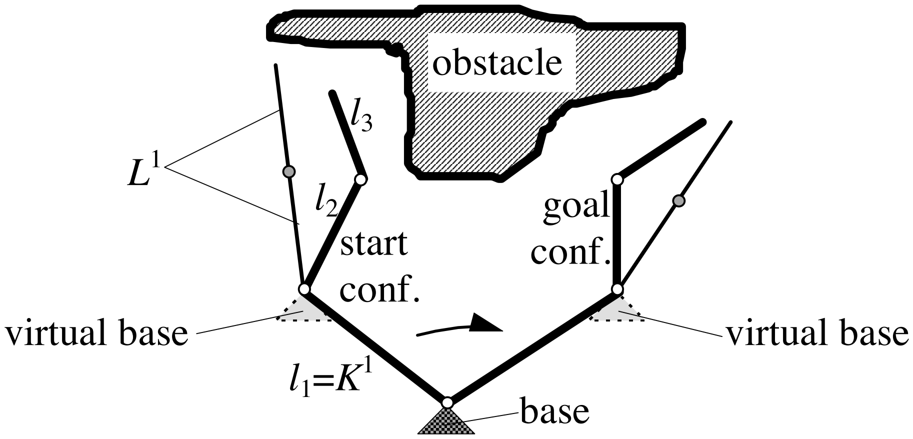


Figure 2.5-9: The complexity of this path planning problem is reduced. Instead of using a 3-dimensional C-space, the planning problem is split into three stages - each stage carried out in a 2-dimensional C-space. First stage: subarm $\boldsymbol{S}^2$ which is made up of links l_2 and l_3 and which is fixed to the left virtual base moves until completely stretched. Second stage: subarm $\boldsymbol{S}^1$ which is made up of links K^1 and L^1 and which is fixed to the main base moves until link K^1 has reached its final configuration (indicated by the arrow) and L^1 has reached a configuration near the desired goal configuration. Third stage: subarm $\boldsymbol{S}^2$ which is now fixed to the right virtual base moves until it reaches the goal configuration.

For example, a 3-link manipulator has to move from a start configuration to a goal configuration in a fairly uncluttered environment, as shown in Figure 2.5-9. The path planning process can be divided into three subprocesses. At first, planning is carried out for link l_2 and link l_3. The C-space patterns for the subarm $\boldsymbol{S}^2$ containing link l_2 and link l_3 can be retrieved from the appropriate look-up table (see Section 2.5). Joint parameters q_2 and q_3 change starting from their initial configuration until an intermediate configuration where q_3 is equal to zero is reached thereby avoiding collision with the obstacle(s). At this stage of planning, subarm $\boldsymbol{S}^2$ is fully stretched. Path planning for the simplified manipulator with q_3 being fixed can be carried out until parameter q_1 is at its desired goal configuration and parameter q_2 is near its desired goal configuration. The C-space patterns due to collisions between the simplified manipulator and obstacle points are stored in the list of subarm $\boldsymbol{S}^1$. After the goal configuration of q_1 is reached, parameters q_2 and q_3 are changed until they reach their final configurations. To calculate the C-space patterns for the final part of the planning, the list of subarm $\boldsymbol{S}^2$ can be utilised again.

Obviously, to construct the second link of subarm $\boldsymbol{S}^1$, link L^1, as a straight line which is comprised of l_2 and l_3 with the configuration of the connecting joint fixed at zero, is not necessarily the best approach to tackle this planning problem. Alternatively, the configuration of the joint which connects l_2 with l_3 could be set to $2\cdot\pi$. Then, l_2 and l_3 would overlap and link L_1 would be represented by a short straight line. (Note, this works only if $|l_3| \leq |l_2|$, because otherwise link l_3 would stick out on the other side of the joint between link l_1 and l_2.) This case could be also treated by the subarm approach, since any link which is smaller than L_1 is a subset of L_1 and is available in the look-up table. However, most physical manipulators can achieve a completely stretched posture, but most of them cannot completely bend without collision between links. Presumably, there is a best configuration to be fixed for each joint of the robot that is least likely to make a collision. This would optimise finding some path, but there may be still better paths which can be only found searching the complete C-space.

This extension to the original approach brings about a reduction in complexity, hence, the calculation is faster and less memory consuming. However, the decoupling of the three-dimensional configuration space into two two-dimensional subspaces means that the conducted path planning is not complete anymore and possible paths might be overlooked (see also [Gupta92]).

2.6 A Radial Basis Function Network for the Workspace to C-space Transformation

This section describes the structure and training of a Radial Basis Function (RBF) network which carries out the workspace to C-space transformation for a two-link planar manipulator with revolute joints (see [Althoefer95, Althoefer95c, Jaitly96b]). The network can be easily adapted to generate C-space patterns for other manipulator types (see Section 2.4.3 and [Althoefer95b]) and can also be integrated into a more complex system to calculate C-space patterns for manipulators with more than two links (see Section 2.5). The network is an implementation of the subarm look-up tables proposed in Section 2.5. It interpolates for those input values which are not part of the training data.

2.6.1 The RBF-Network for the C-space Calculation

The RBF-network used for the workspace to C-space transformation has a single input node through which it receives a scalar value representing the distance between the manipulator base and an obstacle point in workspace. The manipulator here is a two-link stick-like manipulator with revolute joints as shown in Figure 2.4-1. To an input stimulus, the network is trained to respond with a two-dimensional vector representing a discretised C-space pattern.[5]

The RBF-network is used as a universal function approximator [Reignier93]. Like the look-up table presented in Section 2.5, it maps point obstacles into C-space patterns. In contrast to the look-up table, the network interpolates for inputs which are not members of the stored data [Hush93]. This reduces the memory requirements because only a few data points need to be stored to achieve a good performance with a small error. To achieve a similar performance with a look-up table without interpolation capabilities as described in Section 2.5 and [Newman91, Branicky90], more data points have to be stored (see also Section 2.8.1).

The network has $\tilde{s}$ pairs of output units, which represent $\tilde{s}$ configurations (Q_1, Q_2) in the centralised C-space (see Section 2.4.2). To get the actual configuration space of an obstacle point in workspace, the angle θ is added to the Q_1-outputs, and, hence, $(q_1, q_2) = (Q_1, Q_2) + (\theta, 0)$ (see Figure 2.4-1 and Figure 2.4-2). This calculation is done in a separate step.

The network is a two-layer structure whose nodes in the output layers form a linear combination of the basis functions computed in the hidden layer nodes:

[5] Only the nonlinear C-space patterns produced by collisions between link l_2 and obstacle primitives are considered for the training data. Collisions between link l_1 and the obstacle primitive result in straight, vertical stripes which are calculated at a different stage.

$$Q_{m,s} = \sum_{k=1}^{\tilde{k}} w_{m,sk} \cdot f_k(\|d - c_k\|), \tag{2.6-1}$$

where $Q_{m,s}$ represents the angles $Q_{m,s}$, $m \in [1,2]$ at discrete points $s \in [1, 2, ..., \tilde{s}]$ with resolution $\tilde{s}$ (see Section 2.5). The weights in the hidden and the output layer are c_k and $w_{m,sk}$, respectively, while d represents the network input. The shown two-dimensional problem can be easily expanded to incorporate a vector which denotes the width of C-space patterns of a physical arm whose links have a non-zero width (see Section 2.8.1).

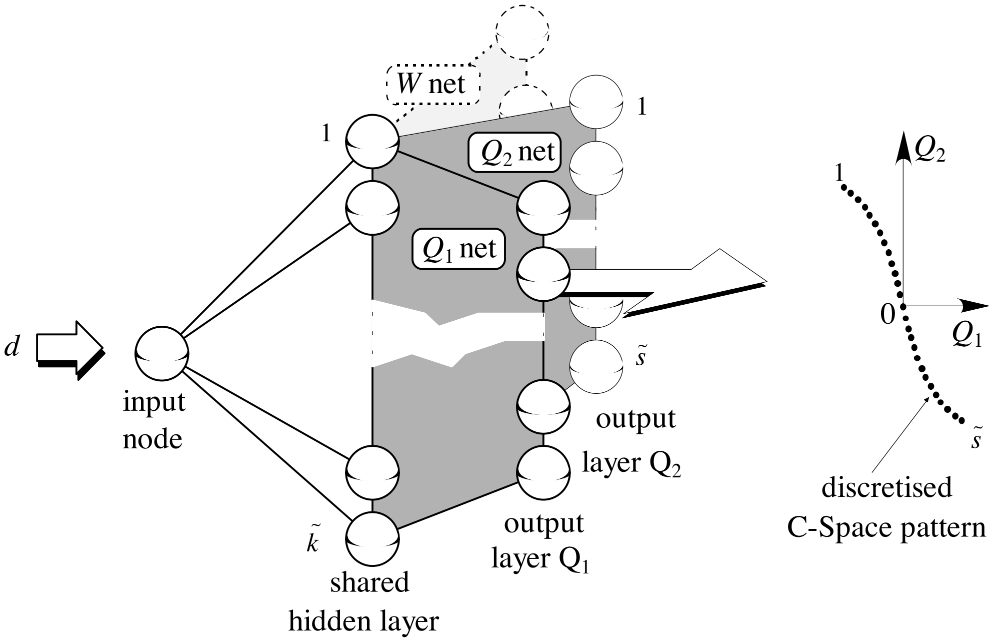


Figure 2.6-1: The network is composed of the Q_1-net and Q_2-net. Each pair of output nodes represents a pair of angles (Q_1, Q_2) of the discretised C-space pattern. The number of nodes in the output layers, $\tilde{s}$, defines the pattern's resolution. An additional subnet, *W*-net, can be added, if the manipulator's links have a certain width. The output of this subnet describes the width of the C-space pattern at the discrete configurations (Q_1, Q_2) (see Section 2.8.1).

In contrast to most other works (i.e.: [Bishop95, Hush93, Musavi92, Miller90]) which commonly suggest the use of a Gaussian function for the hidden layer nodes, a second order dilated B-spline function [Brown94] is employed:

$$f_k(x) = \begin{cases} -\left|\sigma_{l,k}^{-1} \cdot x\right| + 1 & \text{if } x \le \dfrac{1}{\sigma_{l,k}^{-1}} \\ -\left|\sigma_{r,k}^{-1} \cdot x\right| + 1 & \text{if } x > \dfrac{1}{\sigma_{r,k}^{-1}}, \\ 0 & \text{otherwise} \end{cases} \tag{2.6-2}$$

where $\sigma_{l,k} = c_k - c_{k-1}$ and $\sigma_{r,k} = c_{k+1} - c_k$ are the distances to the left and right neighbouring centres, respectively.

As shown in Figure 2.6-2 the basis function $f(\cdot)$ has a triangular shape. Due to the fact that the slopes to either side can be controlled separately, a non-symmetrical receptive field can be obtained. The basis function of each centre has its zero crossings at the closest neighbouring centres, thus, any input value to the network excites one or, at the most, two nodes of the hidden layer. The activity of the nodes is between zero and one, and performs a linear interpolation of those input values which are in-between two centres. The closer the input is to a centre, the larger is the activity of the node. The sum of the activities of excited centres is always one; all other centres output the value zero (see Figure 2.6-2). Generally, the network forms a linear combination of the piece-wise linear basis functions resulting in an overall nonlinear performance. (Compare to the fuzzy-based approach described in Chapter 4.) Experiments with other radial basis functions (Gaussian functions) have been carried out, but the second order B-spline function proved to produce the best results.

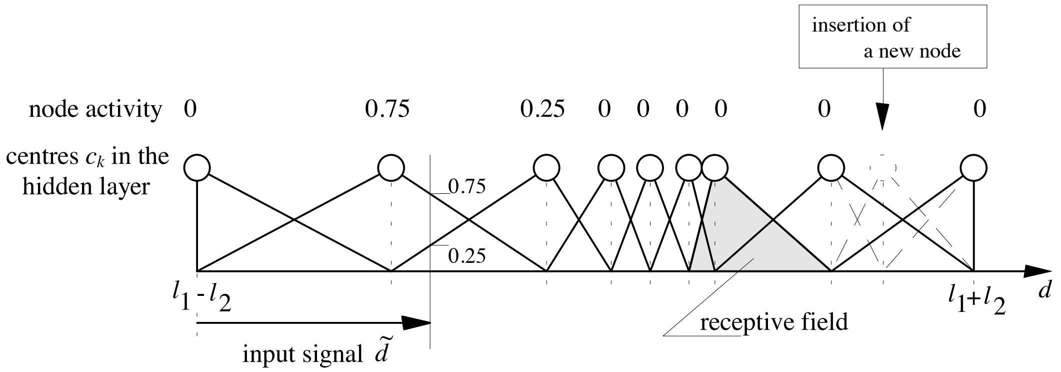


Figure 2.6-2: The dilated B-spline basis functions of the hidden layer nodes distributed over the input space. The receptive field of each node has a triangular shape with zero crossings at neighbouring centres. The figure also shows the insertion of a new node occurring during training (see Section 2.6.2).

2.6.2 The Training of the Network: Insertion of Nodes

The main purpose of the four-phase training method suggested in this section is to reduce the interpolation error between the network response, which is caused by an unknown input stimulus, and the corresponding calculated C-space pattern. This is done by inserting new nodes into the hidden layer wherever the mid-point interpolation error is above a predefined threshold T (see also [Althoefer95c]).

At the start of phase one, the hidden layer of the network comprises a small number of nodes, which, in our experiment, is $\tilde{k} = 5$ (see also Section 2.5). The centres of these nodes, c_k, are set to equidistant d-values in the range $[|l_1|-|l_2|,|l_1|+|l_2|]$. That is, the number of nodes defines the resolution with which the input space is discretised. The resolution of the input space is denoted $\tilde{k}$. In phase two, the weights of the output layer nodes are trained using the least-means-square algorithm [Hush93, Musavi92] to match the C-space patterns for obstacle primitives at distances $d = c_1, c_2, \cdots, c_{\tilde{k}}$.

In phase three, intermediate input values of d (that is, values which had not been used for training yet) are presented to the network. The response of the network to those intermediate inputs is compared to calculated C-space patterns. This test is a simple method to find the approximate maximum interpolation error for intermediate patterns.

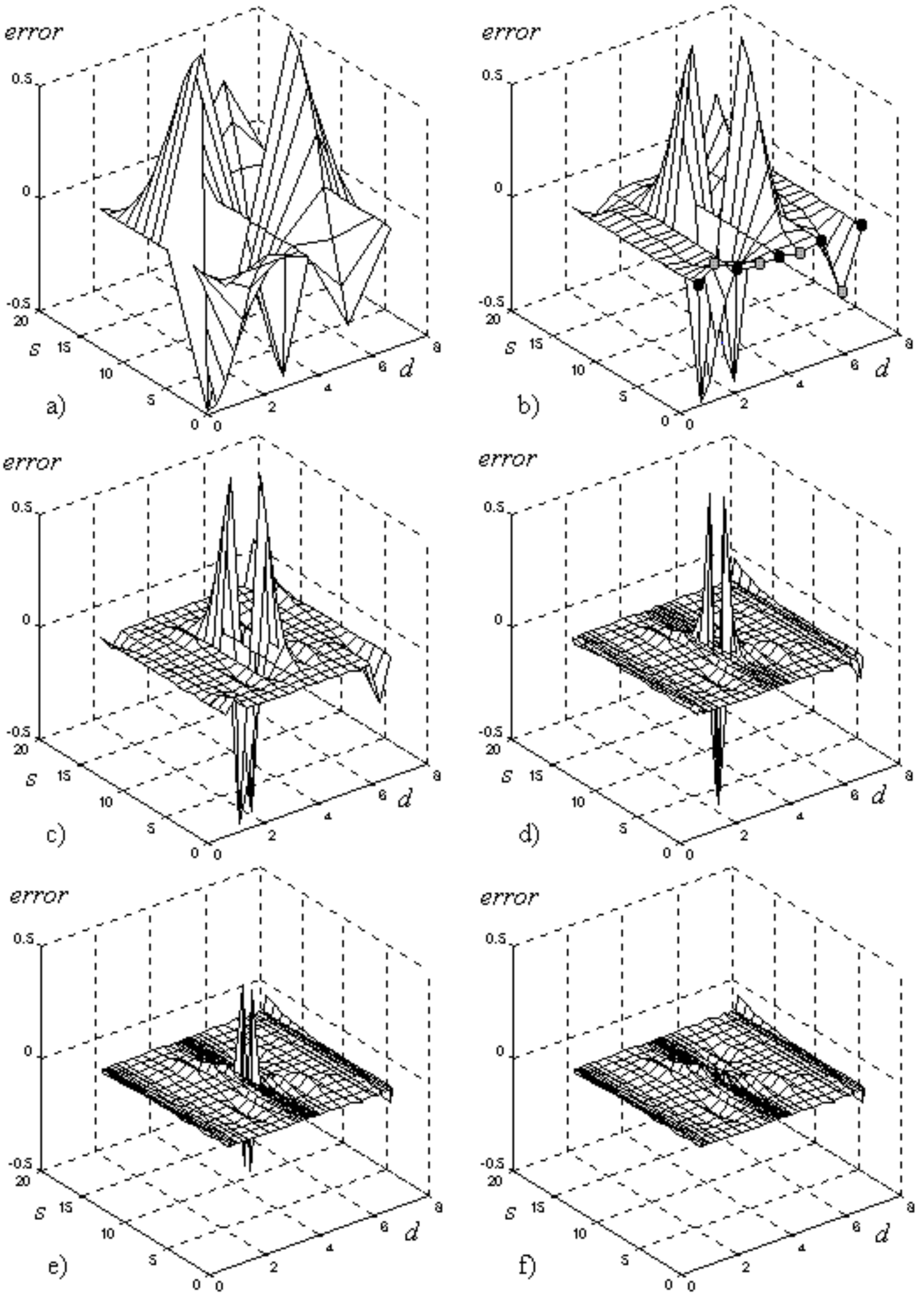


Figure 2.6-3: This figure shows the error of Q_2-net during training. The error of Q_1-net has a similar characteristic. Every second error value along the *d*-axis describes the net error for trained patterns (as indicated by black circles in subfigure b)). These error values are very low and stay low at any iteration. Intermediate error values represent the net error for unknown input stimuli (as indicated by grey circles in subfigure b)). During the training new nodes are inserted wherever the error is above the given threshold *T*. This process increases locally the resolution of the input space along distance *d* and reduces the error for unknown patterns. The number of the output nodes is denoted with *s*. Subfigures a) to f) depict the net error after 1, 2, 3, 5, 7, and 9 iterations, respectively.

In phase four, a new node is inserted into the hidden layer wherever the error between network response and calculated C-space pattern is greater than a predefined threshold T (in this case: $T = 0.1$). The slopes of the new node and the neighbouring nodes are again adjusted in such a way that zero crossings are at the new neighbours (see Figure 2.6-2). The insertion of nodes increases locally the resolution of the input space.

Phases two to four are repeated until the interpolation error is below threshold T everywhere (see Figure 2.6-3).

The experiments conducted have shown that the summed error of the output nodes is reduced rapidly during phase two after only a few tens of iterations [Althoefer95c]. As expected, the responses to trained values of d match the calculated ones very accurately. The responses to intermediate input values show deviations, or peaks, especially when d is close to l_1 where small changes in d lead to big changes in angle q_2 (see Figure 2.6-3). The error peaks diminish as training evolves.

The proposed training algorithm is especially suitable for the training of C-space patterns of physical manipulators. In case of a real manipulator, the C-space patterns are not analytically computed, but measured and stored as discrete values (as described in Section 2.8.1). If after one training cycle is complete the error for a particular d-value does not go below the desired threshold T, a new C-space pattern has to be recorded and the network is trained again. This means that training can be started with only a small set of training pairs. After each training cycle, the network's performance dictates for which values of d, new C-space patterns have to be recorded to reduce the error. Thus, the process of measuring the C-space patterns for the physical manipulator, is reduced to those instances where data actually is needed.

2.7 A Three-link Manipulator

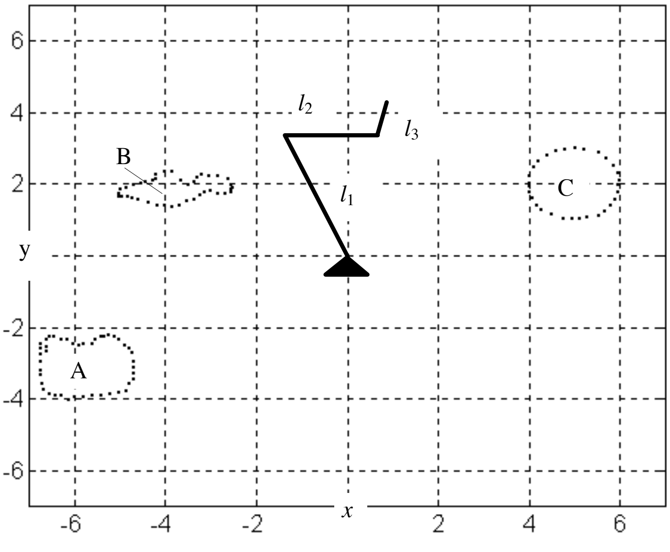


Figure 2.7-1: This figure depicts the planar workspace of a revolute manipulator. Obstacles, "A", "B" and "C", are discretised by pixels. Each pixel represents an obstacle primitive which is transformed into a C-space pattern. The robot manipulator is a simulated arm with three stick-like links. Length of the links: $l_1 = 4$ units, $l_2 = 2$ units, $l_3 = 1$ unit.

This section shows the application of the neural-network-based transformation for a planar stick-like manipulator with revolute joints. Two neural networks have been trained on the C-space patterns of the two subarms $\boldsymbol{S}^1$ and $\boldsymbol{S}^2$ of this manipulator. The construction of the subarms is explained in Figure 2.5-2.

The manipulator is placed in a workspace with three obstacles (see Figure 2.7-1). Each obstacle is comprised of discrete pixels which are separately transformed by the neural network (see Figure 2.6-1). Only those obstacle pixels along the perimeter of the three obstacles, “A”, “B” and “C”, have to be transformed into the configuration space. It has been shown that the C-space patterns that correspond to those pixels which surround an obstacle also engulf the corresponding C-space obstacle (see Section 2.4.1 and [Newman91]).

Although the shown workspace is constructed by hand it approximates well a workspace representation which has been acquired by a CCD-camera (see figures in Section 2.8.3.1). The path planning experiments (Chapter 3) which employ the workspace to C-space transformation method described here showed that the manipulator successfully moved around the workspace obstacles.

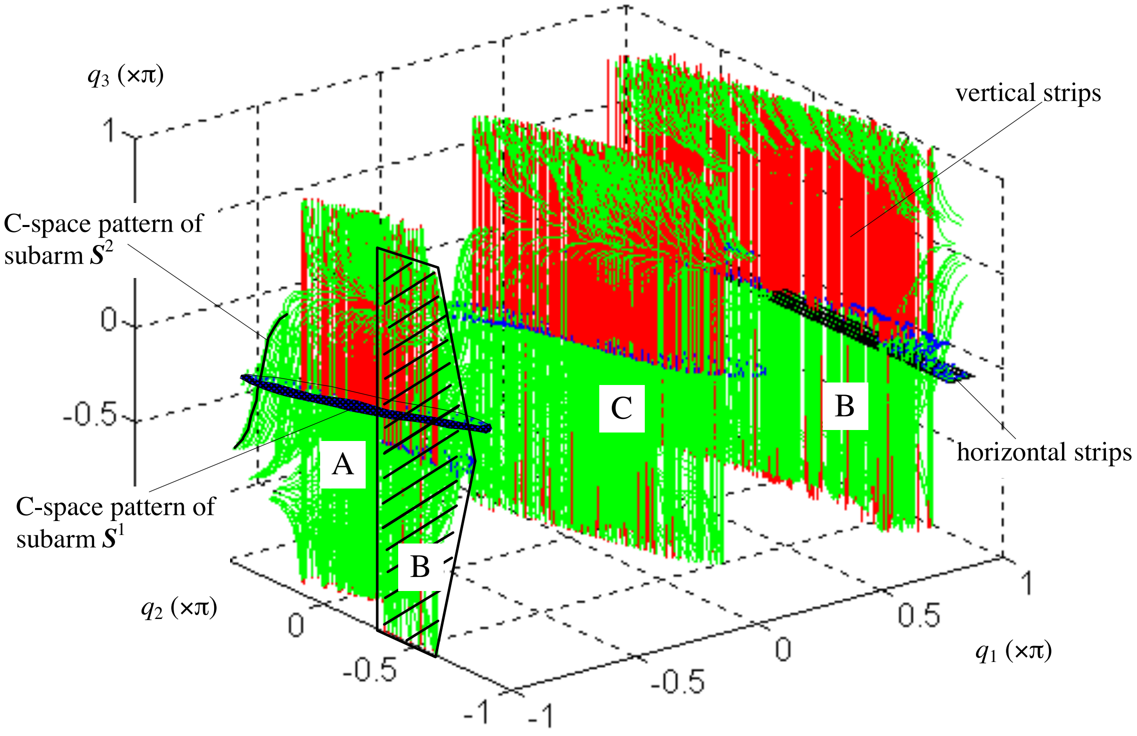


Figure 2.7-2: This figure shows the three-dimensional configuration space of the manipulator in the workspace scenario shown in Figure 2.7-1. Each workspace pixel has been transformed into a C-space pattern. The areas denoted “A”, “B” and “C” represent the C-space obstacles which correspond to the three obstacles in Figure 2.7-1. C-space obstacle “B” continues from the right side to the left side of the diagram due to the modulo-2 property of this C-space (see shaded area in this figure). Horizontal straight strips are caused by collisions of obstacles with link K^1 (which is the first link of subarm $\boldsymbol{S}^1$). Vertical straight strips are caused by collisions of obstacles with link K^2 (which is the first link of subarm $\boldsymbol{S}^2$). The s-shaped patterns due to collisions of subarm $\boldsymbol{S}^2$ “grow” out of the s-shaped C-space patterns of subarm $\boldsymbol{S}^1$, as indicated in the left part of the figure.

2.8 Real-world Applications

2.8.1 C-space Patterns for a Physical Manipulator

So far in this thesis, the workspace to C-space technique has been investigated for manipulators with links that have zero width. The use of point and line robots can be justly criticised as non-real, non-physical. To confirm that the concept developed so far can be applied to a real device, the workspace to C-space transformation technique is applied to a physical manipulator. The manipulator in use is the MA2000 which is an experimental manipulator with three main joints waist, shoulder and elbow and an additional joint which affixes a link called *finger* to the end of the elbow link. A depiction of this manipulator type is presented in Appendix A-1 and Figure 2.8-1. The focus in this section is on the C-space transformations for the shoulder, elbow and finger of the MA 2000 manipulator (see Figure 2.8-1). For the following considerations, the finger joint is fixed to a constant configuration of 0, thus, forming link L^1 of subarm $\boldsymbol{S}^1$. Link K^1 of subarm $\boldsymbol{S}^1$ is link l_1 (see also Section 2.5). C-space patterns for this subarm can be found in Appendix A-2. Subarm $\boldsymbol{S}^2$ consists of links l_1 and l_2. C-space patterns for subarm $\boldsymbol{S}^2$ are depicted in Appendix A-3.

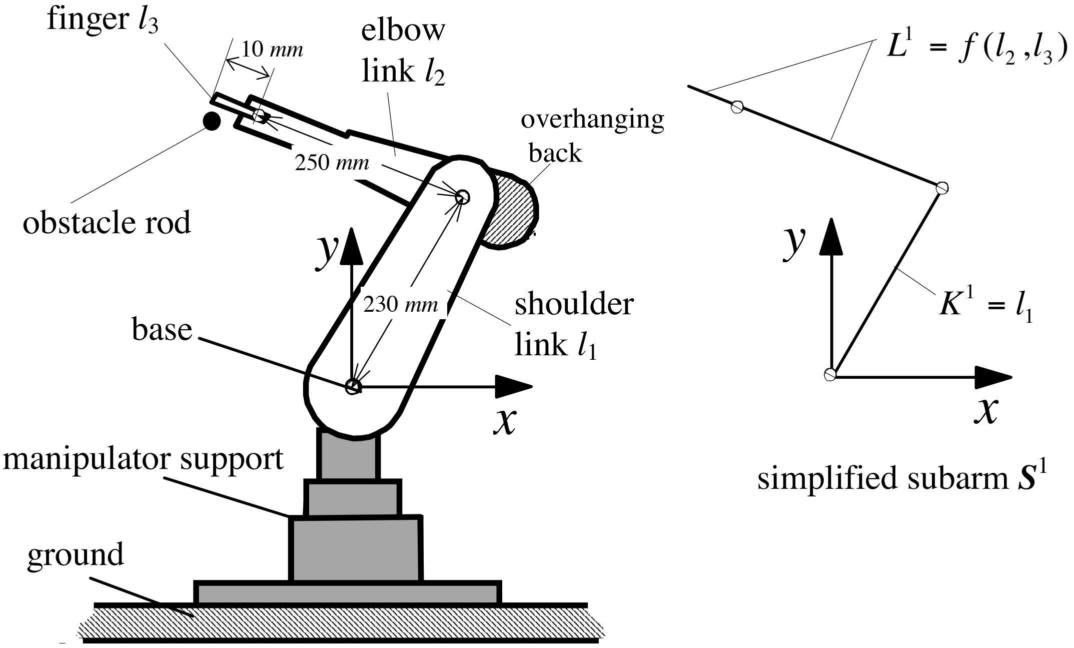


Figure 2.8-1: The experimental manipulator MA2000 equipped with three planar links l_1, l_2, l_3. The origin of the workspace is placed in the centre of the first joint axis (base). The depiction of the MA 2000 is developed from a camera snapshot which has been applied to an edge-detection process and some image enhancing processes.

In order to apply the point obstacle to C-space pattern transformation technique (Sections 2.5 and 2.6) to a physical manipulator, C-space patterns at increments of d have to be generated. For this particular experiment, an apparatus was constructed which allows the acquisition of C-space patterns for point-sized obstacles in varying distances to the manipulator's first joint (workspace origin). The acquisition of a C-space pattern is achieved by manually moving the manipulator in such a way around a thin cylindrical rod (diameter: 3 *mm*) so that there is at all times at least one contact point between the rod and the manipulator. The rod represents the obstacle point. During motion, the manipulator's internal joint sensors are constantly read and the acquired angular values are stored by the connected computer system. The result is a list of elements. Each element denotes all those joint angles which represent an almost-collision with the rod, or in other words, the perimeter of a C-space pattern. The acquisition process is repeated for obstacle rods at different distances from the workspace origin. The collected data is used for the training of the radial-basis-function network (Section 2.6.2). Examples of the acquired C-space patterns are depicted in Figure 2.8-2. A collection of all recorded C-space patterns for this manipulator type can be found in Appendix A-2.

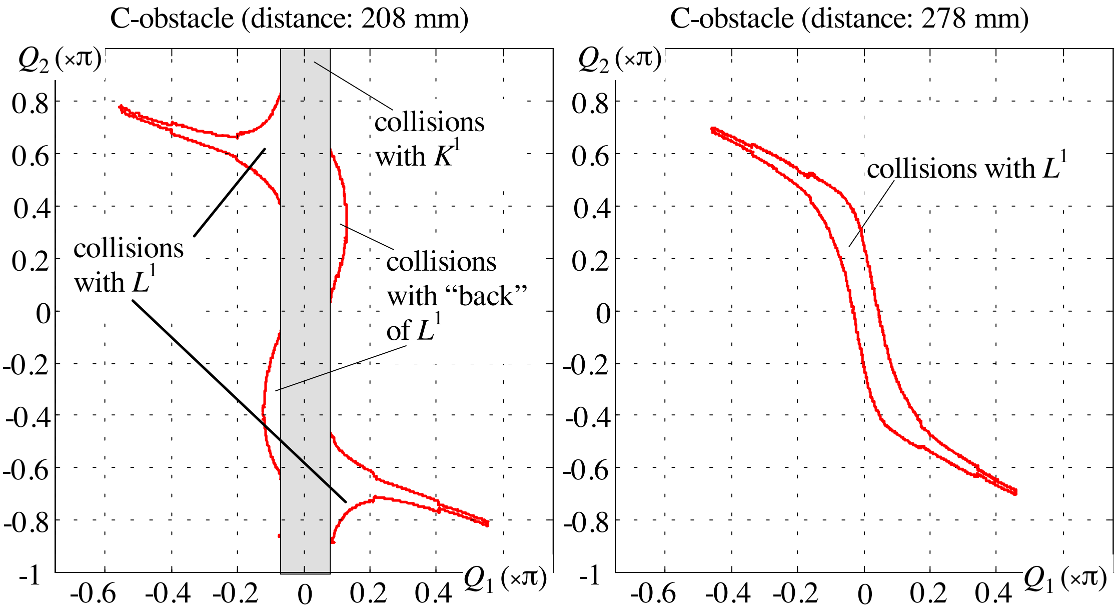


Figure 2.8-2: The measured C-space patterns of links $K^1 = l_1$ and $L^1 = f(l_2/l_3)$ of the MA 2000. In the left figure, the obstacle rod is close to the manipulator's base and, hence, there are collisions with link K^1 (shaded area) as well as link L^1. The right figure shows an example where only link L^1 collides with an obstacle which is out of the reach of link K^1. The perimeters of the C-space patterns are composed of discrete points which are linked up with straight lines in this figure.

The recorded C-space patterns are further processed, as follows. For each C-space pattern, the centre line is computed. The centre line is composed of discrete points which are in the centre (along the Q_2-axis) of those pixels which denote the upper and lower boundary of each C-space pattern. The centre line is described by the two vectors, Q_1 and Q_2. Additionally, for each element of Q_1 and Q_2 the width of the C-space pattern along the Q_2-axis is calculated and stored in a third vector, W (see Figure 2.8-3). The number of elements per vector is a constant value for all C-space pattern (here: $\tilde{s} = 25$). This value represents the resolution of the output space (see Sections 2.5 and 2.6) . The contents of these three vectors were used as training data for the RBF-network (see Section 2.8.3.1). Figure 2.8-3 compares the measured C-space patterns with those reconstructed using the three vectors Q_1, Q_2 and W. As one can see in the right part of the figure, the approximated pattern matches the measured one well. In the left figure, the approximation is good for those parts of the pattern which are outside the marked collision area of link K^1 and the back of L^1. Inside this area, the centre line has no meaning, since all angles Q_1 and Q_2 which are members of this area are treated as forbidden configurations.

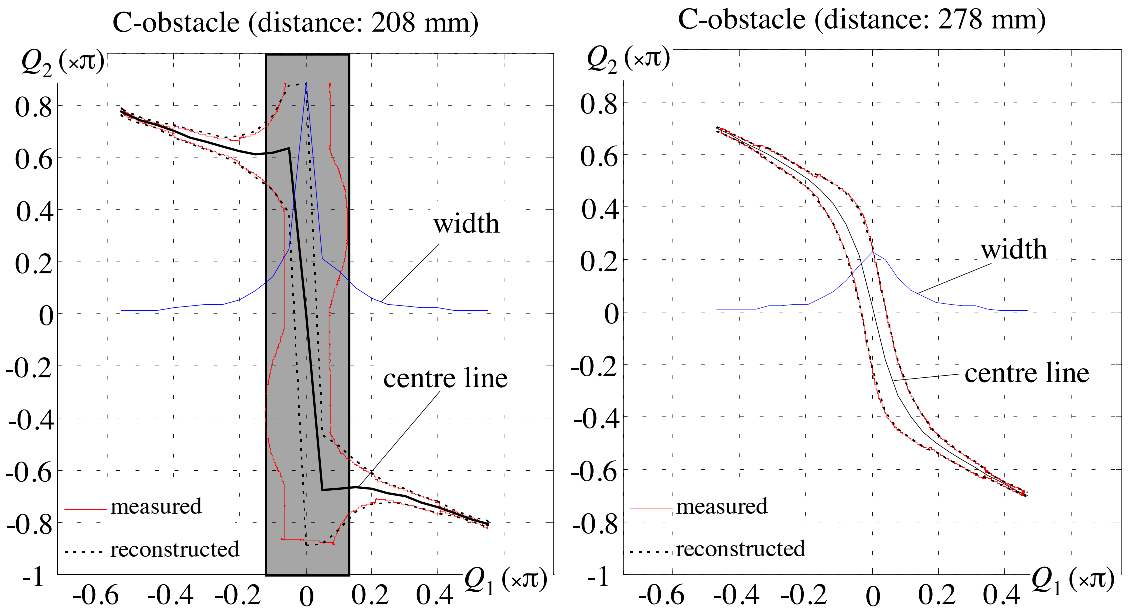


Figure 2.8-3: The measured C-space patterns of links K^1 and L^1 of the MA 2000. Additionally to the recorded patterns, the centre lines are depicted. The grey area in the left figure represents an approximation for the C-space region caused by collisions between obstacle point and link K^1 as well as the back of link L^1. C-space patterns calculated here make use of this approximation. The perimeters of the C-space patterns and the centre lines are composed of discrete points which are linked up with straight lines in this figure. The curve called "width" describes the distance between the centre line and the perimeter of the C-space pattern and is used as a further "pattern" for the training of the network.

Note, the presented technique is particularly suitable for manipulators with links which extend only in one direction from the joints (like for example the stick-like manipulator). The elbow link of the MA 2000 does not satisfy this requirement. However, collisions due to the overhanging "back" of this link (see Figure 2.8-1) can be treated as part of the collision area produced by the shoulder link, as shown in Figure 2.8-3 (left). This approach increases the size of the C-space pattern slightly, and in some rare occasions a possible trajectory close to the shoulder link might not be found. Nonetheless, because forbidden areas are increased, the occurrence of a collision with the corresponding workspace obstacle (here: the rod) is prevented.

The C-space patterns here describe the collisions of links with a small circular obstacle (diameter: 3 *mm*). The size of this circular obstacle is greater than the size of the point obstacles (approx. 0.5×0.5 mm^2) in workspace which are depicted by pixels in the images acquired by the camera used in the experiments of Section 2.8.3.1. Any of those image pixels are transformed into a C-space pattern which actually corresponds to the circular obstacle of *3 mm*. Thus, a safety margin of about 1 *mm* is added.

The tracing of the obstacle rod was repeated for many distances in the range of the manipulator. For each of the recorded C-space patterns, the centre line was extracted. The results are shown in Figure 2.8-4 and Figure 2.8-5. Additionally, the widths of the C-space patterns have been calculated (see Appendix A-2).

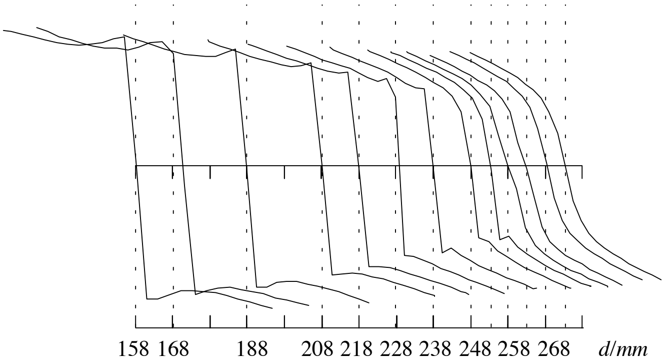


Figure 2.8-4: Centre lines of C-space patterns. The centre lines only represent collisions between link L^1 of the MA 2000 and obstacle points P at distances d. Even though the obstacle points are in the range of link K^1, collisions with this link which produce vertical strips are dealt with at another stage.

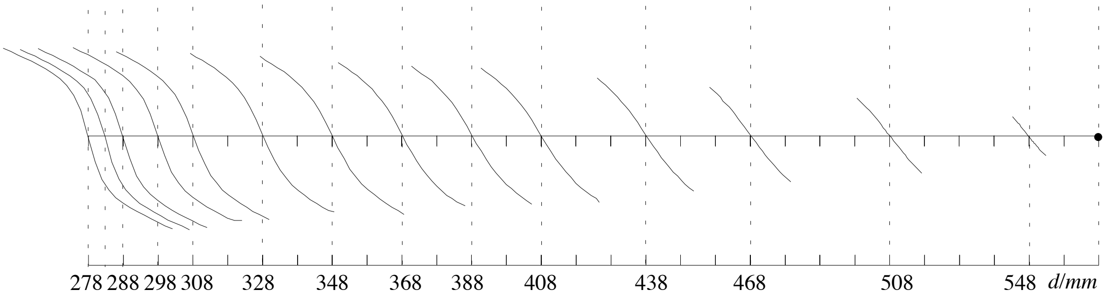


Figure 2.8-5: Centre lines of C-space patterns which are caused by obstacle points outside the range of link K^1.

Similar measurements have been carried out for subarm $\boldsymbol{S}^2$ of the manipulator. Subarm $\boldsymbol{S}^2$ consists of the elbow link and the finger. A complete set of the measured C-space patterns for subarm $\boldsymbol{S}^2$ can be found in the Appendix A-3. The C-space patterns of subarm $\boldsymbol{S}^2$ can be used to construct three-dimensional C-space patterns by "combining" the earlier ones into the C-space patterns of subarm $\boldsymbol{S}^1$, as described in Section 2.5.

The RBF-network (Section 2.6) has been successfully trained on the depicted C-space patterns (Figure 2.8-4 and Figure 2.8-5) and the corresponding "pattern" of width values (see for example Figure 2.8-3). Each such pattern was discretised by $\tilde{s}=25$ values. The number of patterns (see Figure 2.8-4 and Figure 2.8-5) used for training the network was $\tilde{k}=14$ (see also Section 2.6.1). That is, every second C-space pattern of the depicted C-space patterns has been used for training, while the other C-space patterns have been used to test the response of the network to unknown data (Section 2.6.2). The error of the network was in the worst case far below the given threshold $T=0.1$. The memory storage of the network weights required a surprisingly small amount of 12 *kB* only. Branicky *et al.* reports memory requirements of 123 *kB* to store the database for C-space patterns of point obstacles [Branicky90]. They do not report about the error of their technique for unknown obstacle points.

The network-based transformation technique has been used to produce the configuration space depicted in Figure 2.8-6 and Figure 2.8-11. The timing considerations in Section 2.8.2 are based on the network-based transformation of workspace obstacle points into C-space patterns for the MA 2000.

2.8.2 Timing Considerations

Configuration spaces produced by multiple computations of the workspace to C-space transformation technique (Section 2.6) are used as input for the path planner described in Chapter 3. To do so, such a configuration space is loaded into a square map (Figure 2.8-6). Those map nodes which denote collisions (grey areas in Figure 2.8-6) are set to one constant value, while nodes which denote free space (white areas in Figure 2.8-6) are kept on a different value (see Chapter 3). In this way the grid-based path planner which is connected to the map is informed about the manipulator's workspace and can plan a path in the free areas of the configuration space (see Chapter 3 and Section 2.8.3).

Since the generated C-space patterns are composed of a sparse collection of points, it is not sufficient to only mark the corresponding map nodes as forbidden, but also to mark those nodes which lie between. It is essential to construct closed boundaries which enclose the C-space obstacle completely. Gaps in the boundaries could result in paths which wrongly penetrate the C-space obstacles.

Two methods can be thought of. Firstly, if the number of points along the perimeter of the pattern is so high that the distance between two discrete pattern points is smaller than the distance of the grid nodes, no further calculation is necessary and boundary gaps will not occur. Secondly, if, due to the sparse representation of the discrete patterns, the distance between pattern points is larger, an additional algorithm has to be invoked to close the gap. In the programs used here, extra obstacle nodes are introduced along a straight line connecting those nodes which correspond to pattern points.

The C-space boundary (see Figure 2.8-6) is overlaid in a separate process. It describes the perimeter of the manipulator's workspace and is independent of dynamic changes in the workspace. The C-space boundary has been constructed by moving the manipulator in an empty workspace. During the manipulator's motion, its joints' parameters were continuously recorded. Collisions between links, between links and manipulator support as well as between links and work bench define the boundary.

The experiments have been carried out on an IBM-compatible PC with a Pentium90. All programs have been written and executed in the software package MATLAB 4.0. The used version of MATLAB interprets the program code. Equivalent programs executed in compiled form are reported to be 100 to 200 times faster (see [Matlab96] and Chapter 3).

Using the suggested workspace to C-space transformation technique (Section 2.6), the speed of transformation is approximately a linear function of the number of points to be transformed [Newman91]. The experiments showed that the transformation time varies between 0.01 and 0.06 sec./point. The higher of the two times occurred in workspace scenarios where due to collisions of link K^1 vertical C-space strips had to be calculated in addition.

The time needed to store the calculated configuration space into a discrete grid turned out to be long compared to the time needed for the workspace to C-space transformation (see pg. 45). The storing time is proportional to the number of transformed obstacle points and, increases linearly with the square-root of the number of grid nodes. For example, in a 50×50 grid, in a 150×150 grid and in a 400×400 grid, it took 0.125 sec., 0.199 sec. and 0.3575 sec., respectively, to store one C-space pattern of one obstacle point. Recently conducted experiments indicate that by improving the program code the execution time for placing patterns in the grid can be reduced by a factor ten.

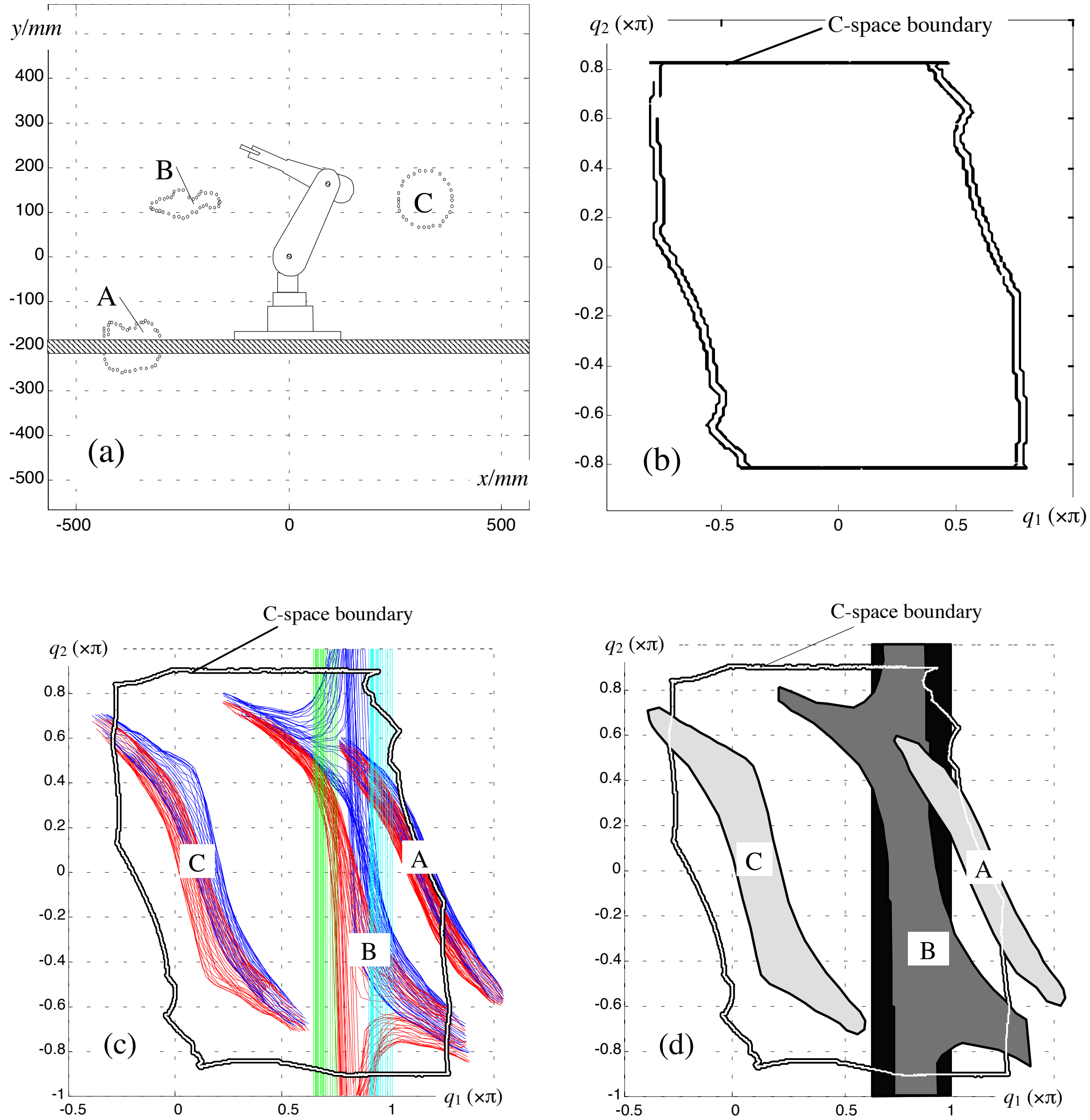


Figure 2.8-6: (a) The MA 2000 in a workspace with work bench, manipulator support and three obstacles. (b) The C-space boundary which describes collisions between links, between links and manipulator support as well as between links and work bench. (c) The C-space patterns. (d) Schematic depiction of the C-space map; the grey and the black areas denote obstacle regions.

The presented technique is especially suitable to transform changes in the environment which are for example due to a few moving obstacles in an otherwise static environment. Instead of recalculating the configuration space for the entire workspace, only those moving obstacles have to be transformed. The C-space obstacles corresponding to the workspace obstacles in Figure 2.8-6 were computed in 1.7 seconds, while the time to store them in a 150×150-grid took 26.14 seconds (see also pg. 45). The shown workspace obstacles comprised a total of 116 individual obstacle points. In this example the static environment is the C-space boundary which is mapped into the grid in a separate step. A comparison with the results achieved in Chapter 3, Section 3.6 showed that the time needed for planning a path in the 150×150-grid is around 90 seconds. Hence, the workspace to C-space transformation technique is well suited as an input to the resistive-grid-based path planner.

Assuming a realistic acceleration by a factor 100, when running the compiled version of the program, the time to transform an obstacle point and placing the corresponding C-space pattern in a 150×150-grid would be around 0.002 seconds. The C-space obstacles corresponding to all workspace obstacles in Figure 2.8-6 could be carried out in about 0.25 seconds. Thus, the configuration space which corresponds to a workspace with three moving obstacles could be refreshed at around 4 *Hz*.

It is difficult to compare results of different researchers, because the transformation process, especially with regards to a real-world problem, depends on many factors, such as computing power, programming language, program details, manipulator type, grid resolution, etc. Often not all this data is made available in a publication. Even if this is the case, the same conditions cannot be reproduced due to differences in available equipment. Newman and Branicky report that the update of the configuration space for a single moving obstacle primitive was executed in the worst case in 0.03 seconds [Newman91, Branicky90]. This computation was carried out on a 16-bit computer (MC68020) running at 16.7 *MHz*. Their configuration space had a resolution of 128×128 [Newman91]. The experiments described here were conducted on an IBM-compatible PC with a Pentium90, which is a 32-bit processor. The transformation of a single moving obstacle point into a similarly sized grid (150×150 nodes) can be done in only 0.002 sec (assuming the acceleration factor 100 as explained in the above paragraph). This means, the workspace to C-space transformation for a real manipulator can be carried out 15 times faster.

The above values are only approximate and shall give an idea on the speed of the proposed technique. Presumably, the execution time can be drastically reduced employing advanced and especially adapted programming techniques. Furthermore, time-critical parts of the program can be coded in assembler.

The suggested technique would show its power if implemented in a parallel computer system. The most obvious approach would be to make available an individual processing unit for every image pixel to be transformed. This would make the transformation time independent of the number of obstacle points. In this aspect, the technique here is superior to any other method where each transformation is applied to complete workspace obstacles, and, thus, cannot easily divided into parallel routines [Lozano87, Latombe91]. The implementation of the look-up list as a neural network implies the possibility of further parallelisation. Every network node can be substituted by a separate processor. This would improve the transformation time per obstacle point. This aspect is especially worth considering, due to the on-going development in the area of hardware-implemented neural networks [Roska96, Koch96, Beiu96, El-Mousa96].

2.8.3 A Real-World Planning System

This section suggests a complete system to solve the FindSpace and FindPath problems (Section 2.2) for a manipulator [Jaitly96b]. The system is capable of planning a trajectory for a real two-link manipulator in a workspace cluttered with obstacles. In a first step, a description of the manipulator's workspace which is acquired by means of a CCD camera is transformed into a configuration space representation (FindSpace). In a second step, the configuration space is loaded into a discrete map in which path planning is carried out

(FindPath). The system combines the techniques presented in this chapter and in Chapter 3. An overview of the system which constructs the C-space is depicted in Figure 2.8-7.

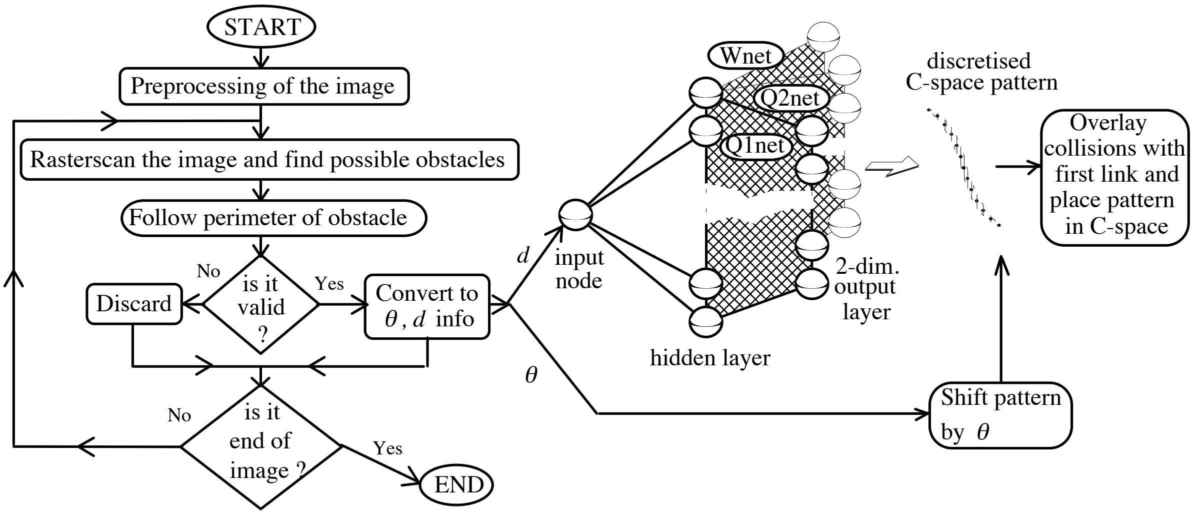

START
Preprocessing of the image
Rasterscan the image and find possible obstacles
Follow perimeter of obstacle
No
is it valid ?
Yes
Discard
Convert to θ, d info
No
is it end of image ?
Yes
END
d
input node
θ
Wnet
Q2net
Q1net
hidden layer
2-dim. output layer
discretised C-space pattern
Overlay collisions with first link and place pattern in C-space
Shift pattern by θ

Figure 2.8-7: Flowchart of image pre-processing and neural network.

The emphasis of this section is on the acquisition of the workspace representation and its communication with the C-space transformation process [Althoefer95c, Jaitly96b]. The core of this process is the RBF neural network (described in 2.6) which is trained to react to a stimulus with a C-space pattern (see also [Althoefer95c]). This stimulus is a scalar representing the distance between obstacle point and manipulator. The real-world experiments show that the transformation of all obstacle points into the corresponding C-space patterns produces a configuration space in which path planning can be carried out successfully [Althoefer95c].

2.8.3.1 Image Processing

The image of the workspace of a planar manipulator with two revolute joints (angular parameters: q_1, q_2) is acquired by a CCD camera which is positioned perpendicular to the workspace (Figure 2.8-8). The obstacles are extracted from the background and discretised into a set of pixels employing a sequence of image processing steps (for details see [Jaitly96b].

After grabbing the image by a CCD camera (Figure 2.8-8) the image is smoothed using a lowpass spatial filter. This removes random, uncorrelated noise from the image. In the following image enhancing steps, edge information is extracted (Figure 2.8-9 (a)), and the image is thresholded to binarize it and separate the background from the manipulator and all obstacles present. The image of the manipulator, whose position is known, is automatically set to zero and hence removed from the image (Figure 2.8-9 (b)). A thinning algorithm is used to produce a silhouette representation of all obstacles present. Furthermore, all remaining “white speckles”, due to noise, are eliminated using a noise removing algorithm [Jaitly96b].

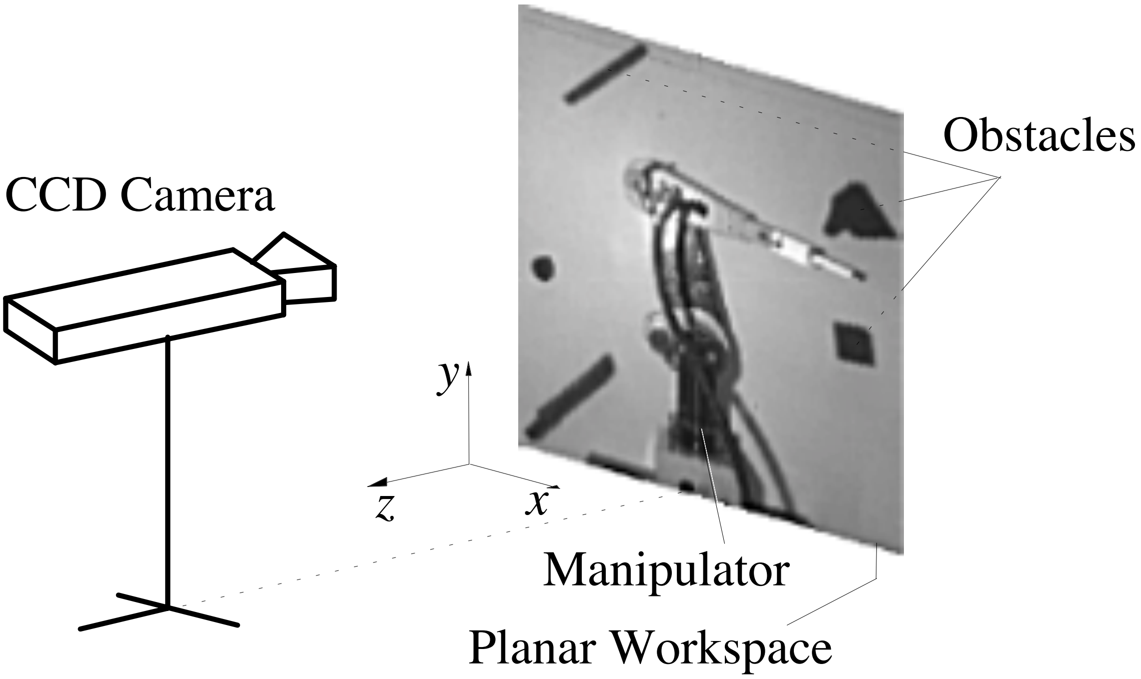


Figure 2.8-8: Manipulator and workspace viewed by the CCD camera.

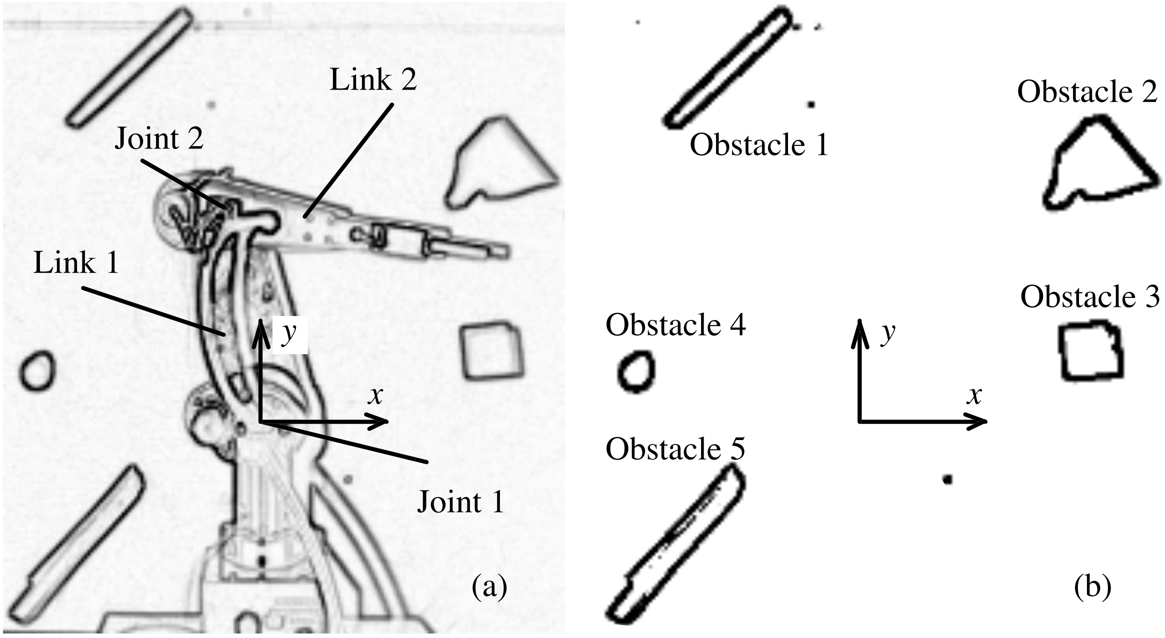


Figure 2.8-9: (a) Edge information of image in Figure 2.8-8, (b) thresholded image with manipulator removed. The origin of the workspace is set to be in the centre of the first joint.

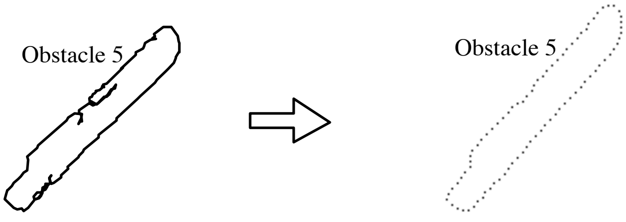


Figure 2.8-10: Result of the fast edge following algorithm. The obstacle perimeters are extracted and divided into equidistant pixels.

Then, a fast edge following process using heuristics is applied to the image to extract obstacle perimeters. Details of this method can be found in [Martelli76, Jaitly96b]. After this process each obstacle is represented by equidistant points on the perimeter of the obstacle. Figure 2.8-10 shows the equidistant points representation of the perimeter of obstacle 5. Each of these points can be interpreted as an obstacle pixel.

The aforementioned image processing took approximately 2-3 seconds for the five obstacles shown in Figure 2.8-9. Most of the time was spent on the thinning of the obstacles (carried out on a serial computer), while the other processes which were carried out on a specialised imaging Transputer board (with a frame grabber, a 2D-hardware convolver and two frame buffers on it) were completed in a fraction of a second [Jaitly96c].

The distance between manipulator and pixel is calculated and individually fed to the neural network (see following section and [Althoefer95c]). The distance between two obstacle points in workspace (corresponding to two pixels in the image) is smaller than the diameter of the obstacle rod which has been used for the generation of C-space patterns (Section 2.8.1). This assures that although each obstacle is represented by discrete pixels, the corresponding C-space patterns overlap sufficiently to form a complete C-space obstacle. If the distance between obstacle points is to high other techniques have to be involved which assure that C-space obstacles with no gaps are constructed (see for example Chapter 3, Section 3.5).

2.8.3.2 Input to the Radial-Basis-Function Network

The RBF network, as described in Section 2.6, is trained to react to the distance value derived from the image processing stage of the system with the corresponding configuration space pattern. The network's output is three-dimensional: two vectors describe the (Q_1, Q_2)-configurations of the C-space pattern, while the third vector describes the width of the C-space pattern at each of these configurations. The third vector incorporates the links' width which make the C-space patterns expand from s-shaped lines to s-shaped areas. These patterns describe collisions between pixels and the second link. Collisions of the first link result in straight vertical strips and are simply overlaid. Since, the network output is only based on the distance of a pixel, the pattern is to be shifted by the angular position which the pixel has in relation to the x-axis to determine its final location in configuration space (see [Althoefer95c, Newman91, Meyer88] and Section 2.4.2). Owing to the interpolation capabilities of the net, unknown patterns are constructed with a low error. The union of the perimeter pixels transforms into the discrete boundary of the corresponding C-space obstacle and multiple network operations are carried out to transform complex obstacles [Newman91, Althoefer95c and references therein].

The discrete C-space depicted in Figure 2.8-11 (a) corresponds to the workspace scenario shown in Figure 2.8-9 (b). The C-space patterns computed by the neural network have been placed into the discrete C-space map by linking up the points of the pattern with straight lines, as described in Section 2.8.2. The map containing the configuration space acts as input to a path planner (see Figure 2.8-11). The trajectory planned by the resistive grid has been applied to a model of the manipulator to depict the functioning of the complete system (see also Chapter 3). The complete transformation process including the image pre-processing is depicted in Figure 2.8-7.

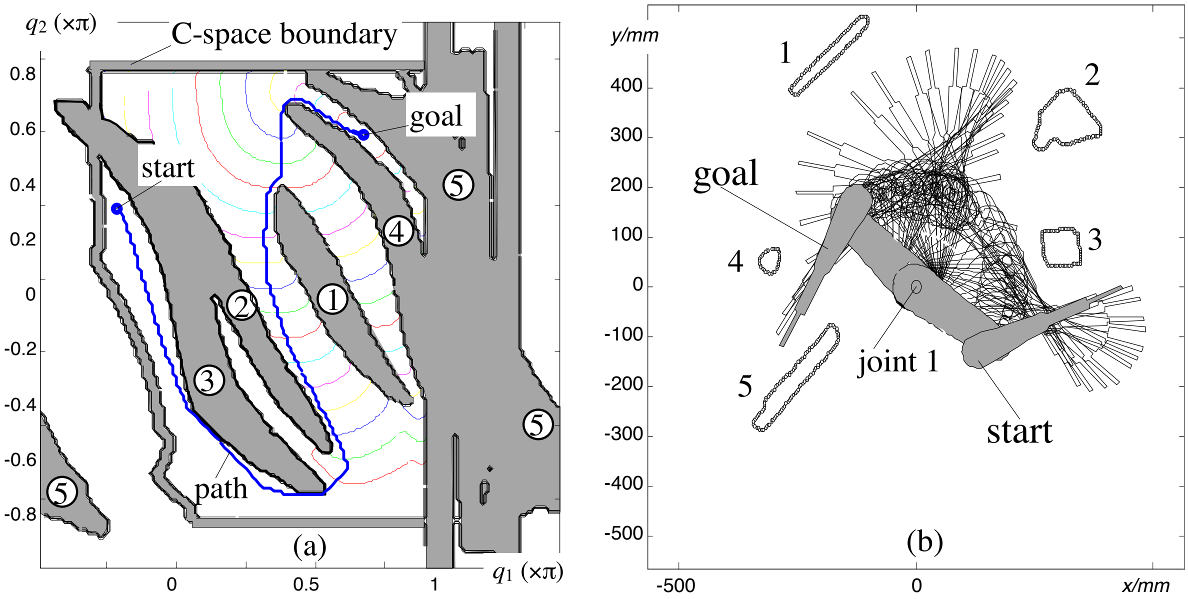


Figure 2.8-11: (a) The grey areas are the C-space obstacles which correspond to the workspace obstacles shown in Figure 2.8-9 (b). The union of the C-space obstacles represents the “forbidden region” not allowed to be entered by the point robot. To demonstrate the purpose of the suggested system, a path has been planned in C-space. The employed planner is based on the neuro-resistive grid. The grid size is 150×150 nodes. For further details, refer to Chapter 3. (b) The path values found in C-space (Figure 2.8-11 (a)) are applied to the manipulator. Each fifth step along the trajectory has been depicted.

2.9 Summary

Path planning in configuration space of a manipulator presupposes the transformation from workspace into configuration space. This chapter presents a neural-network-based technique to transform the representation of single obstacle points into their C-space counterparts named C-space patterns. Multiple of these transformations allow the construction of the configuration space for complex workspace scenarios. The transformation technique has been developed for two-link, stick-like arms which is explained as a fundamental structure of more complex manipulator types.

It has been shown that the technique can be applied to manipulators with different kind of joint types, such as revolute joints, prismatic joints as well as a mixture of the two. The usage of the configuration space for path planning is especially a sensible approach for manipulators with a low degree of freedom. Although this chapter presents a general method which can be used to compute the configuration space for *n*-link manipulators, the dimensionality problem will hamper an implementation for manipulators with a high degree of freedom. Nevertheless, an expansion of the proposed technique for manipulators with more than two links can be easily implemented either in a recursive fashion or in a parallel computer system.

The experimental section of this chapter shows that the proposed technique can be applied to real manipulators. The construction of the C-space patterns for two-link subarms of the experimental manipulator, MA 2000, has been demonstrated. Moreover, the feasibility of the technique as part of a complete path planning system for a real-world problem has been proved.

The radial-basis-function neural network used for the transformation process learns the highly non-linear mapping between obstacle points and C-space patterns. A training technique was introduced which adds new network nodes where necessary. The interpolating capabilities of the network allow the construction of those C-space patterns, which occur in response to unknown input stimuli, with only a small error. The training pairs used consisted, on the one hand, of scalar values representing the distance between manipulator and obstacle point and, on the other hand, of three vectors describing the corresponding C-space pattern. Two vectors represent the discrete centre line of each pattern, while the elements of the third one describe the width of the pattern. The memory requirements to store the weights of the network are particularly small. The network's output can be used as an input for the resistive-grid-based planning strategy proposed in Chapter 3. The network provides a discrete C-space pattern with a resolution of 25 separate configurations. This resolution was sufficient for all the experiments conducted in Chapter 3.

Chapter 3

A Neuro-Resistive Grid for Path Planning

This chapter describes the functioning of a neural-based resistive grid and its application to manipulator path planning (Section 3.1). Its relation to other work is summarised (Section 3.2). A novel update algorithm which allows a fast computation of an activity distribution in the grid providing collision-free paths has been proposed (Section 3.5). This fast update algorithm makes use of a resistive-grid-based approach especially suitable for path planning with strong real-time constraints. The neuro-resistive grid has been successfully applied to real-world manipulators (see Figure 3.6-1 and Section 3.6.1) as well as to simulated manipulators (Section 3.6.2). An extensive comparison to other planning and search techniques shows that the new algorithm is superior in many manipulator path planning applications (Section 3.7). The chapter concludes with a summary of the main aspects of the grid-based planning technique (Section 3.8).

3.1 Problem Definition and Overview of the Algorithm

In today's industrial production processes, path planning is still in its infancy. Most processes are predefined and exact trajectories are provided which explicitly describe any part of the manipulator's motion required for the task to be carried out. The standard procedure is to move the manipulator along the desired trajectory and to record the joint positions which are measured by the manipulator's internal sensors at each time step. Then during operation, the recorded sequence of joint positions is replayed. This basic path planning method is especially useful, when the same motion pattern has to be carried out many times, as it is the case with the assemblage or welding of production goods (as for example cars, electronic equipment, etc.) which passes by the manipulator on a conveyor belt. Since this kind of manipulator system only accommodates internal sensors, it is not aware of the environment and cannot react to possible, but unintentional changes. Thus, such a production process has to be planned precisely and objects to be handled by the manipulator have to appear always at exactly the same position prior to the execution of the programmed motion. A slight deviation of the predefined object positions or manipulator trajectory can lead to damage of the goods as well as the manipulator.

To improve the industrial manufacturing process but especially to open up new areas of applications, manipulators with a more autonomous behaviour are needed. New areas are for example: manipulation of hazardous material, support for disabled people, automated cleaning, painting tasks, etc. In those cases, it is often not possible to predict the exact location of all objects involved in the imagined task. Then, a path planning strategy should be invoked which computes a feasible path shortly before the task is executed. An important means for such a planner is its ability to acquire information about the environment.

External sensors, such as vision-based systems, ultra-sonic sensors or laser scanners, can be used to accomplish this task. Since the acquisition of environmental data is not in centre of the attention of this thesis, it is assumed that a pre-processing step generates a suitable presentation of the workspace (see [Jaitly96b, Latombe91, Siemiatkowska94, Lozano87]). Sensoric equipment produces imprecise, vague or erroneous data (see for example [Latombe91, Lee96, Indyk94]). However, in this chapter, it is assumed that the data acquired by the sensors is exact, thus, it is assumed that the robot moves in an environment where the obstacle positions are known.

A basic problem that often occurs in manipulator path planning is to move the manipulator from an initial position to a goal position. At the goal position the manipulator is usually supposed to carry out a task, as for example to pick up an object, release an object from the gripper, affix the object in the gripper to another object, set a welding point etc. After the task is finished, the present position can be considered as an initial position for a subsequent planning task. In contrast to other planning tasks which involve the tracking of a path (for example path welding), in this thesis the exact trajectory of the path from the initial to the goal position is considered a secondary aspect. However, one important constraint is influential on the construction of the path: the presence of obstacles. If there is an obstacle or a set of obstacles in the workspace, the path planner must assure that the constructed path leads the manipulator through obstacle-free regions. Although the aspect of planning paths which are collision-free is in the centre of this chapter, the proposed up-date algorithm has been developed to produce solutions in short time. The attention in this chapter is not on finding the shortest path between a given start and goal location. Furthermore, aspects which evolve around the dynamics of manipulators are not treated here.

Thus, the centre of attention will be on planning strategies that are based on the kinematic path planning problem which can be described as follows: find a path that leads the robot from an initial configuration to a goal configuration avoiding any collision between the robot and obstacles in the robot's reach, and, furthermore, indicate if such a path cannot be found by the planning strategy [Latombe91].

The path planning strategy presented here is based on a neural implementation of a resistive grid. Resistive grid approaches commonly make use of the *configuration space* (*C-space*) in which each of the robot's configuration is represented by a point in a co-ordinate system spanned by the robot's parameters (see for example [Bugmann95, Althoefer95e, Newman91, Latombe91, Lozano87], Chapter 2 and Section 3.6). Thus, path planning for a complex robot structure is converted to path planning for a single point. While the robot is shrunk to a point, workspace obstacles transform into forbidden regions in configuration space, also called *C-space obstacles*. In most of the experiments described in this chapter, the workspace obstacles have been transformed into their configuration space representation using the method described in Chapter 2 (see also [Latombe91, Newman91, Branicky90]). The resistive grid generates a discrete harmonic function over the free areas of the C-Space. On this function an ascent along the maximum gradient can be performed to guide the *robot point* around obstacles towards the highest potential, the goal configuration.[1]

Simulations and real-world experiments have been carried out to show the feasibility of the proposed approach. The main emphasis was on the use of the resistive grid based method as a path planner for the major links of a planar robotic manipulator in varying environments. Some planning tasks make it necessary that initial and goal configuration are computed from given end effector positions using inverse kinematics. This problem has not been discussed in this chapter; in all cases it was assumed that the manipulator's initial and goal configuration were known.

[1] It should be apparent that $\min[f(x)] = -\max[-f(x)]$. Hence one can restrict one's concerns to either maximisation or minimisation problems. In this thesis, the goal state is always clamped to the maximum value, while the start state as well as any state which is not the goal one are on a lower value. Thus, any path is found by ascending along the potential's gradient.

The neuro-resistive grid has been also successfully applied to autonomous agents in two-dimensional ([Bugmann94, Bugmann95]) and three-dimensional workspaces (see Section 3.6.3).

3.2 Related Work

3.2.1 Resistive Grids for Path Planning

Only recently, resistive grids were discovered as a tool to solve the path planning problem in robotics [Connolly90, Bugmann95, Tarassenko91, Kim91]. To use a resistive grid as a path planning tool for robots has been firstly suggested by Connolly *et al.* [Connolly90, Connolly93]. In their papers, they employ a resistive grid to compute harmonic functions which are solutions to Laplace's equation (see Eq. (3.4-14)). Given certain boundary conditions, harmonic functions have no local extrema over the area of interest, and are therefore well suited for global planning tasks.

The work of Connolly *et al.* is based on a discrete configuration space of a robot. The grid is a collection of nodes which are laterally connected via resistors. Each grid node represents one unique configuration. The node which represents the goal configuration and those nodes which represent the forbidden or obstacle regions are clamped to two differing potentials. Thus, goal node and obstacle nodes represent the unconnected boundary of the grid. This kind of boundary condition is named the Dirichlet boundary condition (see Section 3.4.4). The grid, described by Connolly *et al.*, is emulated on a serial computer. The distribution of the potential over the unclamped nodes has been computed using the Gauss-Seidel update method [Connolly90]. Connolly *et al.* show how -after the potential distribution due to an external input representing goal and obstacles has reached its converged state- a path can be found by following the steepest gradient. This path leads from any point in the obstacle-free area to the goal node [Connolly90].

In response to the papers of Connolly *et al.*, Tarassenko and his colleagues published their experiences with and thoughts about resistive grids [Tarassenko91]. Their main point of criticism is related to Connolly's choice of boundary condition. In contrast to the use of the Dirichlet boundary condition, where the current enters the grid at the goal node and leaves it via the nodes which correspond to obstacles, the Neumann boundary condition dictates that the current enters via the goal node, but leaves via the start node, while obstacles are represented by regions which are insulated from the rest of the state space (see Section 3.4.4). Tarassenko *et al.* show that by using the Dirichlet boundary condition, the grid potential decreases exponentially as the distance to the goal increases, while a grid due to Neumann's boundary condition experiences a mainly linear decrease of the potential [Tarassenko91]. A more detailed description of the two boundary conditions will follow in Section 3.4.4.

Apart from computer simulations, researchers have also constructed VLSI implementations of the resistive grid and similar networks [Tarassenko91, Marshall94, Mead88, Roska96]. Those fundamental experiments are promising, since a hardware grid can generate a harmonic function in very short time. The necessary time is in the order of micro seconds [Roska96, Koch96]. Since a VLSI-grid allows a very fast construction of a potential distribution, the path planning time would mainly depend on the speed of constructing the C-space representation, feeding this representation into the grid as well as reading the found trajectory back. These pre- and post-processing operations are usually carried out on digital computers and tend to have much longer processing times.

Today's achievements in the field of hardware grids are still very limited. The developed grids are only two-dimensional (see also [Tarassenko91, Marshall94, Mead88, Roska96, Koch96] and Section 3.2.3). With the ongoing advances in chip integration, three-dimensional VLSI grids (or two-dimensional ones which are interconnected in a three-dimensional fashion (see also [Ritter92])) of reasonable resolution will be available in the future. Global path planning in a three-dimensional grid applied to manipulators with three degrees of freedom or to the three major links of a manipulator with a higher degree of freedom than three will have a strong impact. Main problems in this field are the connections to the (computer) peripherals and the high interconnectivity inside the grid.

3.2.2 The Hopfield Network

The resistive grid is in its structure very similar to the Hopfield network (see Figure 3.2-1). This network is a lattice of nodes which are interconnected via weights. In contrast to the resistive grid where the resistors have only positive values, the weights in the Hopfield network can have positive or negative values exercising an excitatory or inhibitory influence on the lateral nodes. In the original Hopfield network, every node is connected with every other node and these nodes have a binary transfer function. During operation, an external stimulus is presented and the Hopfield network (like the resistive grid) iterates into a final state.

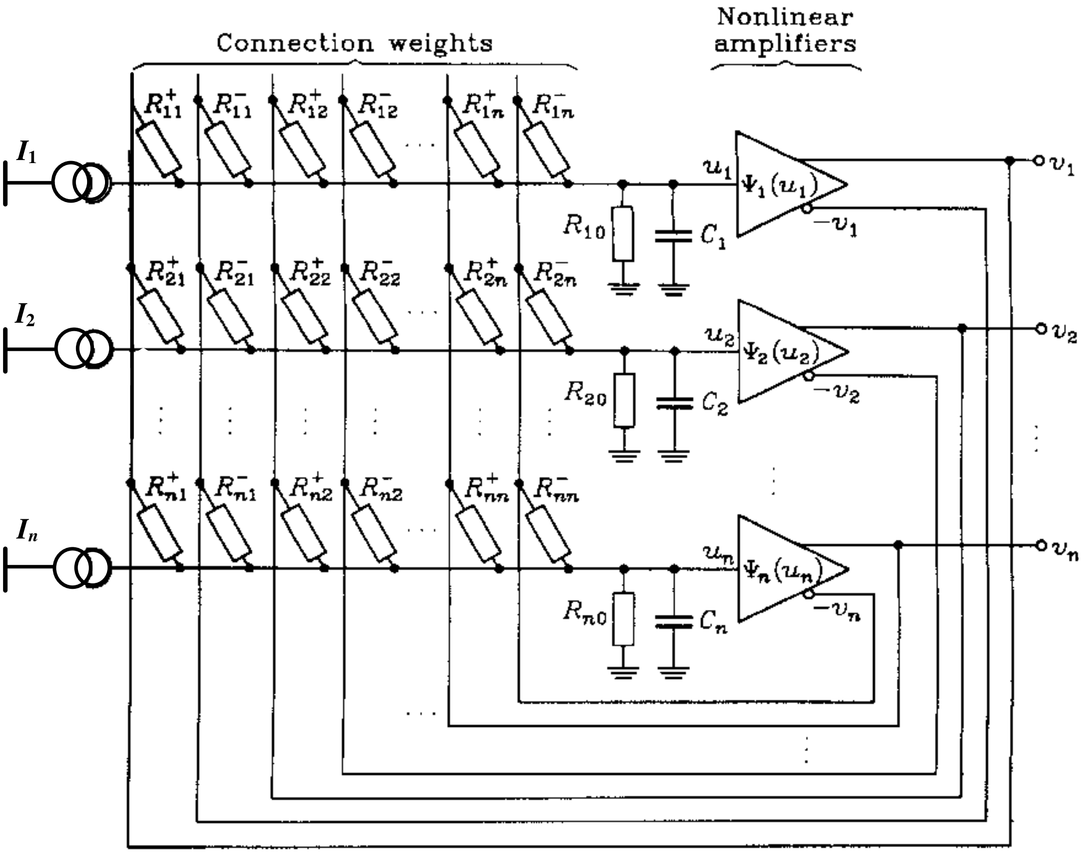


Figure 3.2-1: Structure of a Hopfield network (after [Cichocki94]).

Hopfield applied the magnetic behaviour of spin glasses to neural networks which consist of recurrent binary neurons [Cichocki94]. Most importantly, Hopfield transferred the concept of the energy function which is associated with the behaviour of the spin glass models to his neural network model. The model states that a dipole can only change its spin if this change results in a reduction of the system's energy [Nauck94]. Thus, the model reduces its energy until a minimum is reached. This can be proved investigating the system's *energy function*, also called *Lyapunov function* (see Appendix B, Section 3.4.5 and [Kosko92, Cichocki94, Aiyer90, Kelly90]).

Since Hopfield's first model, many modifications of the original model have been proposed [Cichocki94 and references therein]. Besides the analogue models, time-discrete Hopfield networks have been proposed in the literature [Cichocki94 and references therein, Bose96 and references therein]. The following sections will focus on the time-discrete type, since the experiments and simulations carried out during this research have been done on a serial computer system. Nevertheless, many of the aspects described here also valid for network models implemented on a parallel processing system or in hardware.

3.2.3 Cellular Neural Network

The *cellular neural network* (CNN) is a special case of the Hopfield network. In contrast to the original Hopfield network which is completely interconnected, a cellular neural network is a network structure in which each node is only connected to a local neighbourhood of nodes. The CNN has usually a dimensionality of two and the number of neighbours is normally four or eight [Bose96]. The nodes of a CNN as suggested by Chua *et al.* (see [Bose96 and references therein]) are analogue and all connections between nodes are symmetric. As for the Hopfield network, the stability of the CNN can be proved by employing an appropriately defined Lyapunov function [Bose96, Aiyer90, Kelly90]. A starting condition at each node triggers the response dynamics of the CNN, and it eventually develops into a stable state (see [Bose96], Section 3.4.5 and Appendix B).

Due to its "planar" structure, the CNN is very suitable for VLSI implementations. The two-dimensional CNN can be viewed as a non-linear filter which is already in use to solve different kinds of image-processing problems, for example, noise removal, shape extraction, edge detection and similar [Roska96, Bose96]. Complex image processing tasks have been reported to be carried out in about 1 to 5 μs [Koch96, Roska96].

Computer implementations of the CNN structure have been also used for mobile robot navigation [Siemiatkowska94, Siemiatkowska94b]. The computer-emulated network type proposed by Siemiatkowska is a two-dimensional lattice and represents a point-like mobile robot in its environment. The network is used to plan a path in the robot's vicinity which is scanned by ultra-sonic sensors [Siemiatkowska94b]. Each node or cell in this particular implementation of a two-dimensional CNN is laterally connected to neighbouring cells. The cell which represents the goal location of the robot is considered to be the source for a diffusion process and is clamped to a large positive value. Obstacles are clamped to zero. An update rule is applied to calculate the spreading of the diffusion over the obstacle-free areas of the lattice. Informally, at each iteration, the current cell activity becomes a percentage of the largest activity in the neighbourhood. The iteration process is stopped when either the diffusion has reached the current robot location or all obstacle-free areas have been investigated without reaching the current robot position. When a passable connection between start and goal exists a gradient ascent from start location to the goal location can be performed to generate a path. This method is free of local extrema up to the resolution of the lattice. Kanaya *et al.* proposed a hardware implementation of a CNN which is used to plan paths for mobile robots [Kanaya94].

An approach based on the principles of Hopfield networks has been described by Glasius *et al*. [Glasius94]. Their network is very similar to a cellular neural network as well as the neuro-resistive grid. It consists of neurons which are connected to local neighbours in a lateral fashion. In their paper, the network is applied to solve the path planning for a point-like mobile robot and a "stick-like" two-link manipulator in environments cluttered with obstacles. Goal state and obstacle states are clamped to a positive value and zero, respectively. The used update equation is the same as used for the neuro-resistive grid (see Eq. (3.4-11)). Their paper focuses on the dynamics of Hopfield networks. It shows that the particular neural network converges to one stable final state and therefore is appropriate to solve global planning problems. The presented proof of convergence can be also applied to the neuro-resistive grid (see [Glasisus94] and Appendix B).

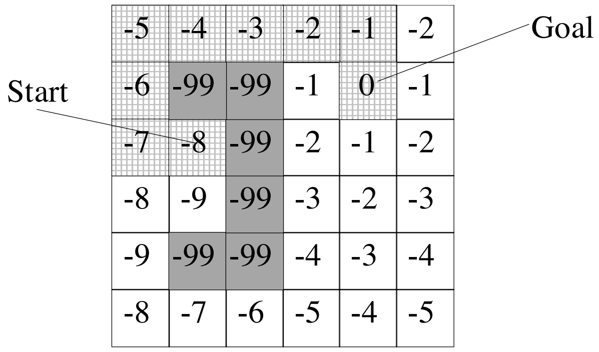
3.2.4 Dynamic Programming

Similar to resistive grid approaches, *dynamic programming* can be considered to be a particular approach to solve different kinds of optimisation problems, such as path planning, graph search, travelling salesman, logistics, equipment replacement and investment optimisation [Cooper81]. The principle of dynamic programming or the *principle of optimality* were first developed and introduced by Bellman [Bellman57]. Dynamic programming is usually applied to optimisation problems in systems which progress through consecutive stages. The task is to find an optimum sequence of stages which connects a given start state with a goal state. The quality of the sequence is reflected by an overall cost (or reward). The sequence with the minimum cost (or maximum reward) is the optimum route (see also footnote on page 54).

Finding the optimum sequences of stages, is a sequential process which is based on a decision to be made at every stage. At each stage the system is said to be in a certain state and depending on this state a decision can be made. Each state is associated with a set of available decisions which depend on the actual state the system is in. A decision carried out results in a change from the actual state the system is in into a new state. This transformation from one state to another is associated with a local cost to be paid.

Employing a dynamic programming approach, usually a look-up table or list is built which contains a cost value for each state of the state space (see Figure 3.2-2). The construction of this look-up table begins at the goal state. This state is special, since once this state is reached, no further transition takes place, therefore, the goal state is also called absorbing state [Cooper81]. Each state which is a neighbouring state to the goal state is associated with a cost. In a next step, states are considered which are neighbours to the neighbours of the goal state. The cost value of such a state is the accumulation (for example a sum or a product) of their local costs and the previously calculated total costs of the states surrounding the goal state. For each state, only the minimum cost is stored in the look-up table. The process of assigning a minimum total cost is continued until every state has been visited. Thus, on completion of the look-up table, each state is assigned with a value which denotes the minimal cost along subsequent states towards the goal state. This description reflects the recursive nature of dynamic programming, since the total cost assigned to any state depends on the total costs assigned to states visited beforehand. In other words [Cooper81]: "An optimal policy has the property that whatever the initial state and the initial decision are, the remaining decisions must constitute an optimal policy with respect to the state which results from the initial decision."

Once the look-up table is built, a sequence with minimum cost from any state to the goal state can be found easily. At each transformation from one state to another the chosen subsequent state should be the one with the minimum total cost. A very good introduction to the principles of dynamic programming can be found in [Cooper81]. For a mathematical treatment of dynamic programming and application examples, refer to [Bellman57, Bellman62].



-5	-4	-3	-2	-1	-2
-6	-99	-99	-1	0	-1
-7	-8	-99	-2	-1	-2
-8	-9	-99	-3	-2	-3
-9	-99	-99	-4	-3	-4
-8	-7	-6	-5	-4	-5

Figure 3.2-2: The figure depicts the cost look-up table for a point-sized robot in a two-dimensional environment. The table is generated by a dynamic programming approach. The total cost assigned to each position (state) is the sum of the local cost (which is always -1 in the obstacle free regions) and the minimum cost of neighbouring states. The value -99 is assigned to the obstacles. This value is more negative than the accumulated cost of any obstacle-free state. Thus, a state representing an obstacle will never be considered as an obstacle-free state. The hatched squares denote a path from the chosen start position to the goal position.

The A*-algorithm which is often used in robotics to solve search graph problems is based on dynamic programming. The main difference of the A*-algorithm in contrast to the general dynamic programming approach is that the A*-algorithm does not assign to every possible state a cost value but terminates as soon as the start state is encountered during the construction of the look-up table. The A*-algorithm will be discussed in further depth in Section 3.7.3.

The aforementioned approaches, namely the resistive grid based approach [Connolly90, Tarassenko91, Bugmann95, Althoefer95e], the Hopfield-network-based approach [Glasius94] and the CNN-based approach [Siemiatkowska94b], which have been suggested for robot path planning can be viewed as particular implementations of dynamic programming. In each case, a topological discrete map is formed which describes the configuration space of a robot in its environment. Each node in the map represents a state of the robot. The transition from one state to another is combined with a weight or cost which represents the Euclidean distance between those two states. In each case, a table or list is generated in where costs accumulate. In the approach chosen by Siemiatkowska, the accumulation of costs is not done in an additive manner but by means of a product rule. In the resistive grid and the Hopfield approach, the activity or total cost of one state is achieved by averaging over the activities of neighbouring states (further explanations will follow in Section 3.4).

Approaches based on dynamic programming commonly construct a look-up table of costs as described above (see also Section 3.7.3). In contrast to this method, resistive grid based approaches and similar approaches often use update methods which are iteratively applied to the nodes representing the states. The iterative procedure is repeated until convergence or some other criterion is satisfied. The activity distribution which is yielded resembles a look-up table of costs (see Section 3.7).

From the observations made, it seems that all described approaches follow the same principles and can be explained with the theory of dynamic programming. However, in the literature all these approaches have been separately treated as completely different and no attempt has been undertaken to show the strong similarities. To further analyse these aspects and to construct a general framework which include all aforementioned approaches seems to be an interesting area of research, but is beyond the scope of this thesis.

3.3 Path Planning in the Configuration Space

Over the last three decades, many path planning strategies were developed. These strategies are usually split into two stages. In the first stage, a graph is constructed which represents the collision-free space or *free space*. Examples include the visibility graph method, the Voronoi diagram, the free way method, the cell decomposition method. In a second stage, the constructed graph is searched for a collision-free path from a start to a goal location. Methods commonly used to accomplish this task are the A*-algorithm (see Section 3.7.3) and other map searching algorithms. These search methods are mostly global planning strategies which find a path if such exists. For a good historical review of motion planning strategies for mobile robots and articulated manipulators, refer to [Latombe91].

All the aforementioned path planning strategies have in common that in their basic form they can be easily applied to a point-sized robot. Hence, with the development of a new tool to represent the robot and its surrounding environment, those strategies gained in importance. This tool was the so-called *configuration space* (*C-space*) whose principles are described in Chapter 2 (see as well [Lozano83, Latombe91]).

Conceptually, the configuration space (C-space) approach by itself represents the robot and workspace, rather than providing a feasible path for the robot. Thus, path planning in configuration space is usually comprised of two stages: the identification of those robot configurations which do not result in a collision, called *FindSpace*, and subsequently the search for a path within this representation which connects start and goal configuration, called *FindPath*. While the first stage was dealt with in Chapter 2, this chapter discusses the problem of finding a path in the constructed configuration space.

The main drawback of employing the C-space for the solution of path planning problems is that the C-space increases exponentially with the robot's degree of freedom. This causes problems in terms of memory and computing time. Nevertheless it has been already shown that paths can be planned for manipulators with six degrees of freedom on a single processor [Ralli96], and with the advent of highly parallel machines this method will gain ground, especially, when discretised C-spaces are used and path planning strategies like the resistive grid are applied to these C-spaces. This is because these strategies are readily implementable and highly prallelisable. In view of the dimensionality problem, many techniques have been developed which make use of the C-space idea, but employ heuristics to reduce the dimensionality. These techniques aim at constructing a C-space representation which does not make use of all possible degrees-of-freedom of the robot. Examples can be found in [Latombe91, Gupta92, Lozano87] (see also Chapter 2). Despite the advantageous reduction of the C-space complexity, those techniques do not necessarily find a path under any possible condition and are therefore not global anymore. However, the resistive-grid-based approach proposed in this chapter can be used as a FindPath strategy in these simplified C-spaces.

3.4 The Neuro-Resistive Grid

3.4.1 Implementation of the Neuro-Resistive Grid

The neuro-resistive grid is a collection of N nodes which are distributed over a state space (see [Bronshtein85]). Thus, the grid can be seen as a discretisation of a state space where each node represents one individual state. In most of the applications investigated in this thesis, the neuro-resistive grid is used to do path planning in a discrete configuration space (*joint space*) of robotic manipulators. The neuro-resistive grid is based on the work described in [Bugmann94, Bugmann95, Bugmann96, Althoefer95d, Althoefer95e]. The neuro-resistive grid has been also applied to path planning for mobile robots (see [Bugmann94, Bugmann95] and Section 3.6.4) as well as the control of the inverted pendulum [Bapi95, D'Cruz96].

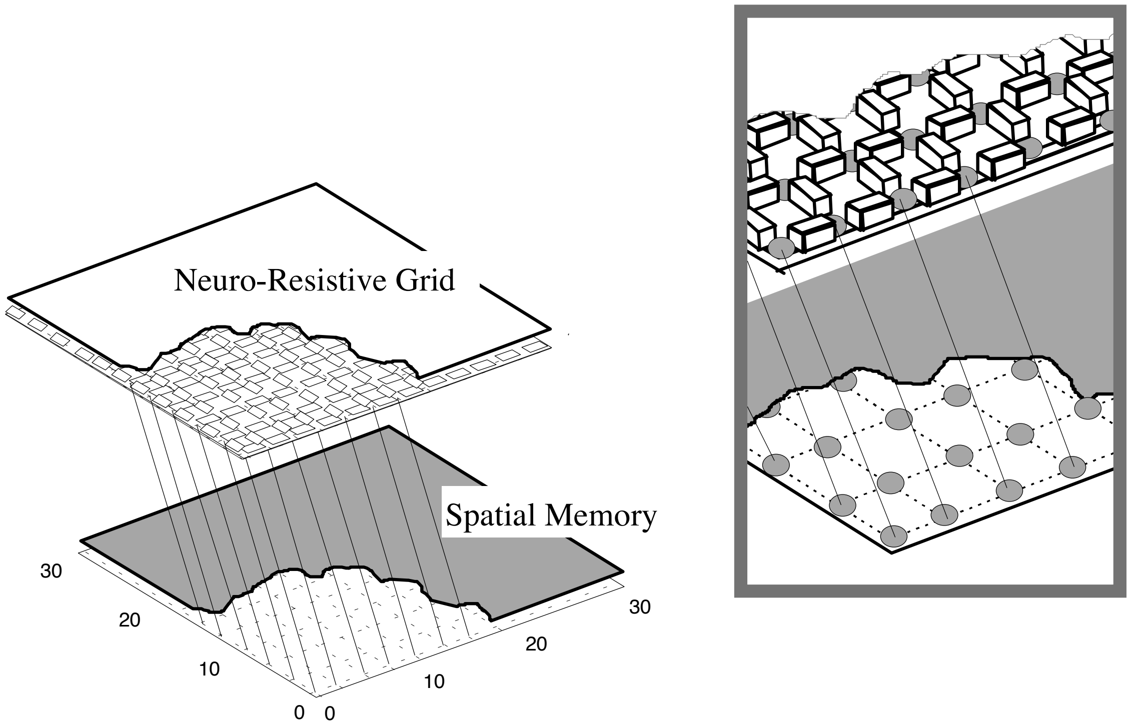


Figure 3.4-1: This figure shows the neural implementation of a 2-dimensional resistive grid. The neuro-resistive grid is one-to-one connected to the layer which contains the spatial memory.

The location of a node in the co-ordinate system of the grid corresponds to one individual robot configuration. Usually, the nodes are arranged in an n-dimensional lattice whose Cartesian co-ordinates describe the joint parameters. The dimensionality n of the grid is the same as the one of the underlying configuration space. The lattice represents a topologically ordered map [Glasius94, Ritter92]. Instead of a Cartesian co-ordinate system, other map types can be used to achieve a discrete and topologically ordered description of the configuration space. Maps which are built by Kohonen networks have been used to construct configuration space representations and can be also used as input to the resistive grid [Glasius94 and references therein, Ritter92, Coolen91].

In the grid, each grid node *i* is connected via resistors to nodes in the local neighbourhood. Although different connectivity schemes are conceivable, the focus in this thesis is on two different schemes. Firstly, each node is connected to the closest horizontal and vertical neighbours (see Figure 3.4-1 and Figure 3.4-3), and secondly, each node is connected to the closest horizontal and vertical as well as the closest diagonal neighbours (see Figure 3.6-5). The latter of the two schemes allows the production of smoother paths, since the diagonal shortcut is also possible. Depending on the chosen scheme each node has either four or eight neighbours in a 2-dimensional grid. Along the edges of the resistive grid, nodes have fewer neighbours. Usually, the resistor values depend on the number of neighbours and, hence, the resistors connected to edge nodes should be appropriately adapted. This aspect is not further investigated here, since in all experiments the configuration space is bounded, that is edge nodes are clamped to zero and are not influenced by their neighbours.

Each node in the grid receives inputs from the lateral neighbours as well as from one node which is situated in the *spatial-memory* layer which is also called *input map* (Figure 3.4-1). The spatial memory is a collection of nodes which merely buffers the current configuration space constellation. Any node which represents a forbidden state (part of a C-space obstacle) has the activity "-1". The node which corresponds to the goal configuration is set to "+1". The rest of the nodes in the spatial memory has activity "0" and describe the C-free space. Those activities which describe goal and obstacles constrain the behaviour of the nodes in the neuro-resistive grid as described in the following (see also [Bugmann95]).

Each node *i* in the resistive grid is a processing unit whose output signal or activity v_i is:

$$v_i = f\left(\sum_{\substack{j=1\\ j\neq i}}^{N} w_{ij} v_j + V_i\right), \tag{3.4-1}$$

where w_{ij} is the weight matrix describing the connection between neighbouring nodes (details: see Section 3.4.2), v_j is the output of node j, f is the activation function of node i, N is the number of nodes in the grid, and V_i is the external input from the spatial-memory layer. If the external signal V_i describes a forbidden state, the output of node i in the resistive grid saturates to "0" and node i acts as a current sink. If the external signal V_i describes a goal, the output of node i in the neuro-resistive grid saturates to "1" and node i acts as a current source. Those nodes in the spatial memory which are set to "0" have no influence on the corresponding nodes in the resistive grid and are called *unclamped nodes*. Goal and obstacle nodes in the grid are called *clamped nodes*.

Any activation function $f(x)$ which satisfy the following conditions can be used: The activation function $f(x)$ has to increase strictly monotonically, saturate at "1" for large positive values of x, saturate at "0" for large negative values of x, and the output must be zero for $x = 0$. Two possible activation functions are (see also Figure 3.4-2):

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ kx & \text{if } 0 \le x \le 1, \\ 1 & \text{if } x > 1 \end{cases} \tag{3.4-2}$$

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ \tanh(kx) & \text{if } x \ge 0 \end{cases} \tag{3.4-3}$$

where k represents the gain of the activation function. Other sigmoidal functions which satisfy the above requirements can be also used [Bugmann95]. Gain k is not allowed to exceed a certain limit to ensure stability and convergence (see Section 3.4.5). The first function will be referred to as the *linear saturating activation function*, and the second one will be referred to as the *tanh-activation function*.

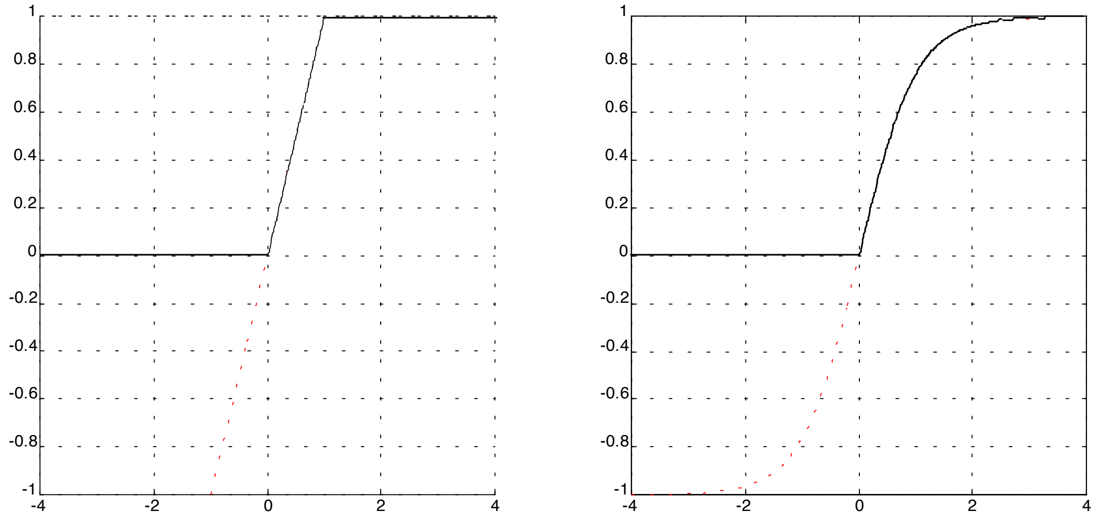


Figure 3.4-2: The linear saturating function (left) and the tanh-function (right). Both function are used as activation functions in the neuro-resistive grid. For details see text.

3.4.2 Functioning of the Resistive Grid

Path planning in the resistive grid can be carried out by applying a voltage at the goal configuration and a different voltage at all those nodes which represent forbidden areas[2]. In the grid, a potential distribution occurs which depends on goal and obstacle configurations.

Kirchhoff's Law can be used to derive the set of equations describing the potential distribution in the resistive grid. The calculation of the activity or the potential of any node i, $i = 1,\ldots,N$ in the resistive grid is based on Kirchhoff's Current Law which states that the sum of currents entering a node i is zero (see also [Cichocki94]:

$$\sum_{\substack{j=1\\ j\neq i}}^{N} i_j + I_i - C_{i0}\frac{\partial v_i}{\partial t} = 0, \qquad (3.4\text{-}4)$$

where i_j, $j = 1,\ldots,N$, is the current flowing from node i to node j, I_i is the external current source, v_i is the potential at node i, and the term $C_{i0}\frac{\partial v_i}{\partial t}$ describes capacitive effects due to the switch-on transient (see Section 3.4.5). Eq. (3.4-4) represents a set of linear equations which can be rewritten using Ohm's Law:

[2] This corresponds to the Dirichlet boundary condition. The differences between the two commonly used boundary conditions, which are named Dirichlet and Neumann boundary condition, are explained at a later stage of this chapter (Section 3.4.4).

$$C_{i0}\frac{\partial v_i}{\partial t}=\sum_{\substack{j=1\\ j\neq i}}^{N}\frac{v_j-v_i}{R_{ij}}+I_i=-v_i\tilde{w}_i+\sum_{\substack{j=1\\ j\neq i}}^{N}v_j\frac{1}{R_{ij}}+I_i, \tag{3.4-5}$$

where v_j is the potential of node j which is connected to node i via resistor R_{ij}, and $\tilde{w}_i=\sum_{\substack{j=1\\ j\neq i}}^{N}\frac{1}{R_{ij}}$ represents the sum of the reciprocal of the resistors connected to node i.

Eq. (3.4-5) can be reorganised as follows:

$$\tau_i\frac{\partial v_i}{\partial t}=\sum_{\substack{j=1\\ j\neq i}}^{N}w_{ij}v_j+V_i-v_i, \tag{3.4-6}$$

where

$$w_{ij}=\frac{1/R_{ij}}{\sum_{\substack{j=1\\ j\neq i}}^{N}\frac{1}{R_{ij}}}=\frac{1/R_{ij}}{\tilde{w}_i} \tag{3.4-7}$$

represents a weight parameter which connects node i with node j, $\tau_i=\frac{C_{i0}}{\tilde{w}_i}$, and $V_i=\frac{I_i}{\tilde{w}_i}$ represents an external voltage supply. A more detailed explanation on the values of weight w_{ij} will be given at a later stage of this section.

While Eq. (3.4-6) describes the analogue model in the time continuous case, the equation system can also be solved in the discrete time domain using the following recursion formula which is also known as Gauss-Seidel iteration or successive relaxation method:

$$v_i^{(k+1)}=v_i^{(k)}+\Delta v_i^{(k)}=v_i^{(k)}+\frac{\Delta t}{\tau_i}[\sum_{\substack{j=1\\ j\neq i}}^{N}w_{ij}v_j^{(k)}+V_i-v_i^{(k)}] \tag{3.4-8}$$

For $\Delta t=\tau_i$, follows:

$$v_i^{(k+1)}=\sum_{\substack{j=1\\ j\neq i}}^{N}w_{ij}\cdot v_j^{(k)}+V_i, \tag{3.4-9}$$

where $v_i^{(k)}=v_i(k\tau)$ with sampling period τ (see [Cichocki94, Reimer86, Connolly90]). Eq. (3.4-9) describes the grid update where every node is updated simultaneously and synchronously. In this case, the index order does not matter. The sequential update rule to calculate the activity distribution in the grid is as follows:

$$v_i^{(k+1)}=\sum_{j=1}^{i-1}w_{ij}\cdot v_j^{(k+1)}+\sum_{j=i+1}^{N}w_{ij}\cdot v_j^{(k)}+V_i. \tag{3.4-10}$$

Here, the update order has influence on the propagation of the activity distribution, since those values which had been already “processed” during the update cycle enter again the equation [Reimer86]. In Eq. (3.4-10), nodes are updated in the order $1,\ldots,N$. Eq. (3.4-10) produces a faster spread of the activity distribution than Eq. (3.4-9). The update order has a strong influence on the speed with which the activity spreads over the grid. This issue which is a central point of this chapter will be discussed in detail in Section 3.5.

Eqs. (3.4-9) and (3.4-10) can be also expressed in matrix form which represents the iteration rule independent of the update order:

$$\mathbf{v}^{(k+1)} = \mathbf{w} \cdot \mathbf{v}^{(k)} + \mathbf{V}, \qquad (3.4\text{-}11)$$

where

$$\mathbf{w} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & \cdots & w_{NN} \end{bmatrix}$$ representing the interconnection matrix,

$\mathbf{v}^{(k)} = \left[v_1^{(k)}, v_2^{(k)}, \ldots, v_N^{(k)}\right]$ and $\mathbf{V} = \left[V_1, V_2, \ldots, V_N\right]$.

The interconnection matrix **w** contains the weights between the nodes of the entire grid. The matrix is symmetric ($w_{ij} = w_{ji}$) and any connection between two nodes is excitatory ($w_{ij} > 0$). The matrix's main diagonal is set to zero ($w_{ij} = 0$), since no feedback from node i to itself is considered. Moreover, a weight w_{ij} is set to zero, if there is no connection between node i and j. Matrix **w** is a general description of the connections in the resistive grid. This description is independent of the dimensionality of the grid. Thus, Eqs. (3.4-9) and (3.4-10) are valid for grids with n dimensions. However, in the programs used for the experiments (see Section 3.6), a neighbourhood template which moves along the grid nodes during the update process defines which nodes are neighbours to the present node i (see Table 3.4-1). Node i is the node whose activity becomes the average of the activities of the nodes defined in the template. Thus, in the used programs the matrix **w** is not computed explicitly.

$w_{k\text{-}1,l\text{-}1}$	$w_{k,l\text{-}1}$	$w_{k+1,l\text{-}1}$
$w_{k\text{-}1,l}$	0	$w_{k+1,l}$
$w_{k\text{-}1,l+1}$	$w_{k,l+1}$	$w_{k+1,l+1}$

Table 3.4-1: Two-dimensional neighbourhood template where each node is connected to eight neighbours. Indices k and l represent the location of node i in the two-dimensional grid. The index increment and decrement by 1 indicate the location of the nearby neighbours.

In the resistive grid as proposed here, the activity of any node is only influenced by a small number of nodes in the near neighbourhood, thus (see as well [Glasius94, Bugmann95])

$$w_{ij} = \begin{cases} \hat{w}_{ij} & \text{if} \quad \hat{d}_{ij} < d \;\wedge\; j \neq i \\ 0 & \text{otherwise} \end{cases}, \qquad (3.4\text{-}12)$$

where weight $\hat{w}_{ij}$ represents an element of the neighbourhood template (see Table 3.4-1). The value of each element depends on the Euclidean distance $\hat{d}_{ij}$ between node i and node j as well as on the number of nodes in the range described by the value d,

$$\hat{w}_{ij} = \frac{\hat{d}_{ij}}{\sum_{j \in N_i} \hat{d}_{ij}}, \tag{3.4-13}$$

where $\hat{N}_i$ represents the set of influential neighbours of node i. The number of nodes which are influential on node i depends on the number of nodes node i is connected to via a resistor (see also Section 3.4.4). Eq. (3.4-13) also assures that the sum of all weights connected to a node is always 1 (see also Section 3.4.5). For example, in a 2-dimensional grid where the nodes are placed in equidistant positions along the two axes and d is chosen to be 1.5, each node i has eight neighbours. For this case, Eq. (3.4-13) produces two different weight values, one for connections between nodes in vertical and horizontal direction and another for connections between nodes in diagonal direction:

$$\hat{w}_{hor\&ver} = \frac{1}{4 \cdot 1 + 4 \cdot \sqrt{2}} = 0.104, \quad \hat{w}_{diag} = \frac{\sqrt{2}}{4 \cdot 1 + 4 \cdot \sqrt{2}} = 0.146.$$

Experiments showed that choosing the average of the two weights, which is 1/8, leads to a similar activity distribution in the grid and produces the same path (see also [Glasius94]).

3.4.3 Harmonic Functions

A resistive grid can be used to approximate the calculation of harmonic functions on a confined, connected region $\Omega \in R^n$. Such a solution can be found on a computer employing an iterative process where at each iteration the potential at each node i in the region Ω is substituted by the average potential of its neighbours (see previous section).

A harmonic function in Ω satisfies Laplace's Equation

$$\Delta\phi = \sum_{k=1}^{n} \frac{\partial^2 \phi}{\partial x_k^2} = 0, \tag{3.4-14}$$

which describes physical phenomena like the distribution of an electrical field in a charge-free space or a current flow through a conductive medium (see [Connolly90, Connolly93, Ramo94]). Eq. (3.4-14) imposes that the electrical field or current is free of any source or sink in the internal region Ω. Thus, maximum and minimum are on $\partial\Omega$ which is the usually unconnected boundary of region Ω. Furthermore, the solution is unique [Ramo94] and depends only on the shape of boundary $\partial\Omega$ [Glasius94]. From the above considerations follows that the resistive grid is well suited to solve the path planning problem in robotics providing always a collision-free path from a start state to a goal state, if such path exists.

In the discrete case, one considers a set of discrete grid nodes which represent the discrete counterpart to the continuous medium [Bronshtein85]. Hence, continuity is assumed from node to node. Each node can be assumed to be situated on the crossing point of parallel and perpendicular lines placed equidistantly in a rectangular co-ordinate system. Other homogeneous grid shapes are possible but not further discussed in this thesis (see [Bronshtein85]). For example in a 2-dimensional homogeneous grid where $v(x_i, y_i)$ represents a discrete regular sampling of ϕ and each node is connected to four neighbours (see also Figure 3.4-3), the first partial derivative of $v(x_i, y_i)$ with respect to x can be calculated (see also [Bugmann95]). Note that in the following equations node distance a has to be sufficiently small to allow the approximations carried out:

$$\frac{\partial \phi(x_i, y_j)}{\partial x} = \lim_{a \to 0} \frac{v(x_{i+1}, y_j) - v(x_i, y_j)}{a} \approx \frac{v(x_{i+1}, y_j) - v(x_i, y_j)}{a} \qquad (3.4\text{-}15)$$

and

$$\frac{\partial \phi(x_i, y_j)}{\partial x} = \lim_{a \to 0} \frac{v(x_i, y_j) - v(x_{i-1}, y_j)}{a} \approx \frac{v(x_i, y_j) - v(x_{i-1}, y_j)}{a}, \qquad (3.4\text{-}16)$$

where a is the node distance in x- and y-direction. The second derivative can be derived by differentiating Eqs. (3.4-15) and (3.4-16),

$$\frac{\partial^2 \phi(x_i, y_j)}{\partial x^2} \approx \frac{v(x_{i+1}, y_j) - 2v(x_i, y_j) + v(x_{i-1}, y_j)}{a^2}. \qquad (3.4\text{-}17)$$

The second partial derivative with respect to y can be obtained in the same way. The sum of the two partial derivatives gives (compare to Eq. (3.4-14)):

$$\begin{aligned} \Delta\phi(x_i, y_j) &= \frac{\partial^2 \phi(x_i, y_j)}{\partial x^2} + \frac{\partial^2 \phi(x_i, y_j)}{\partial y^2} = 0 \\ &\approx \frac{v(x_{i+1}, y_j) + v(x_{i-1}, y_j) + v(x_i, y_{j+1}) + v(x_i, y_{j-1}) - 4v(x_i, y_j)}{a^2}. \end{aligned} \qquad (3.4\text{-}18)$$

The same result can be achieved, if a Taylor expansion is carried out (see for example [Connolly90, Noble64]). Eq. (3.4-18) can be rearranged into:

$$v(x_i, y_j) \approx \frac{v(x_{i+1}, y_j) + v(x_{i-1}, y_j) + v(x_i, y_{j+1}) + v(x_i, y_{j-1})}{4a^2}. \qquad (3.4\text{-}19)$$

If distance a has the value 1, Eq. (3.4-19) goes over into Eq. (3.4-1).

The above equations hold only for topologically ordered grids where nodes are only connected to nearby neighbours and any connection does not intersect with any other connection (see also Section 3.5.4). Thus, these grids can be seen as a discrete representation of an Euclidean space. Step size a might vary as a function of space but has to be sufficiently small to allow the approximations carried out in Eqs. (3.4-15), (3.4-16) and (3.4-17). Note that Laplace's equation is only valid when the grid (whether the physical or the computer implemented one) has settled into its final potential or activity distribution. In the computer implemented resistive grid the activity of each node changes with each iteration until no further change occurs in the limit of the computer's precision. Then the grid has converged. For this limiting case only, Laplace's equation is satisfied and no local extremum can occur in the internal region of $\boldsymbol{\Omega}$.

If the update process is interrupted before the activity for all nodes has stabilised, the distribution is not a solution to Laplace's equation but to Poisson's equation according to which the second derivative in space is equal to a time depending function (see [Bugmann95, Noble64]). In the unconverged grid, the statements made above do not necessarily hold, thus, uniqueness and the min-max principle cannot be presumed. For further details see Section 3.5.3.

3.4.4 Boundary Conditions - Dirichlet vs. Neumann

Solutions of Laplace's equation have to satisfy conditions on the boundary $\partial\Omega$ of region Ω (see [Bronshtein85]). The most commonly used boundary conditions are the Dirichlet- and the Neumann-boundary conditions.

If the Dirichlet-boundary condition is used, the boundary $\partial\Omega$ is set to some fixed values. This approach has been adopted for the neuro-resistive grid described here. In the grid these values are fixed potentials. For example obstacle regions are set to 0V and the goal is set to 1V (see Figure 3.4-3). Other boundary values are possible; the only condition is that the values for goal and obstacle have to be different. Obviously if goal and obstacle boundary values were equal, an activity distribution with values different from goal and obstacle values could not develop, since this would violate Laplace's equation which demands that minimum and maximum have to be on the borders. In case of the Dirichlet boundary condition, obstacles can be seen as *zero-voltage borders* (see Figure 3.4-3).

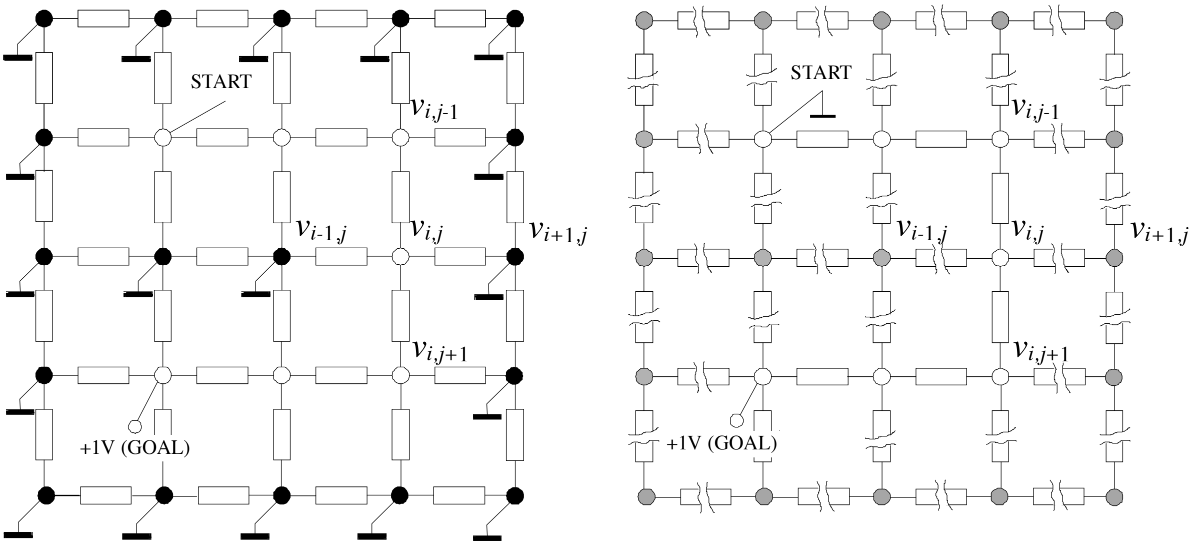


Figure 3.4-3: This figure shows the structure of a resistive grid due to the Dirichlet boundary condition (left) and the Neumann boundary condition (right). In this example the connectivity scheme is chosen where each node is connected to four neighbouring nodes.

In case of the Neumann boundary condition, the boundary $\partial\Omega$ is equal to defined derivatives of the values which occur in the internal region. In the resistive grid, this corresponds to insulating the obstacle regions from the internal region. Thus, obstacles become *zero-current borders*. If the Neumann boundary condition is applied, the start and goal nodes are clamped to two different potentials (here: 0V and 1V, respectively) (see Figure 3.4-3).

Although the same update algorithm is used for iteratively calculating the potential distribution in a resistive grid employing any of the two boundary conditions, there is a subtle difference. The difference can be seen by analysing Eq. (3.4-13). For the Dirichlet boundary condition, the output of obstacle node i is clamped to zero. The weights w_{ij} are independent of the obstacle configuration. For the Neumann boundary condition the weight parameters have always to be adapted to the obstacle constellation surrounding the node to be updated. Obstacle nodes which surround node *i* are isolated from the latter. Thus, there is no resistor connection between node *i* and its obstacle neighbours. This in turn means, that the number of nodes having influence on node *i* is reduced and the weight factor $\hat{w}_{ij}$ in Eq. (3.4-13) changes.

Most experiments described in this thesis have been based on a resistive grid applying Dirichlet's boundary condition. A comparison in computing time showed that a Dirichlet grid is about 1.5 times faster than a grid using the Neumann boundary condition to produce a path. This is due to the fact that the weight factor has to be recalculated at every iteration, if the Neumann boundary condition is applied, while it stays constant for the Dirichlet boundary condition (see also [Bugmann95]). This difference in speed was found using the standard division commands as they are provided by higher programming languages to divide the sum of neighbouring activities by the number of neighbours. The calculations in a Dirichlet grid could be further accelerated, if fast shift operations were used to achieve the division by four or eight.

A further disadvantage of the Neumann condition is that the grid has to be recalculated, if the start position changes. In contrast, the grid's potential distribution due to Dirichlet's boundary condition can be reused to compute paths from any starting position in the grid to the goal as long as obstacles and goal stay unchanged.

It has been reported that the use of the Dirichlet boundary condition can fail when long paths have to be calculated because the precision of the computer system might not be sufficient [Tarassenko91]. Using the Dirichlet boundary condition, the potential decreases mainly exponentially as the distance to the goal increases. This effect is especially observed in long, narrow corridors. The required computing precision is proportional to the corridor's length divided by its width [Tarassenko91]. Thus, the computer's precision must be high to be able to recognise the marginal potential difference between two adjacent nodes which are in a narrow corridor and far from the goal node. In case the Neumann boundary condition is applied, the potential decay appears to be linear over wide regions. Only near goal and start can a stronger decrease be observed (see Figure 3.4-5 and [Tarassenko91]).

Tarassenko *et al.* suggest the use of the Neumann boundary condition to avoid the exponential decay of the potential (see [Tarassenko91]). Especially, in a hardware implementation the possible precision of the grid is limited by the analogue gates which read the potential at the nodes. Today's precision of gates in analogue computers for an affordable large scale implementation is between 5 to 8 bits [Kramer96]. Thus, for a hardware implementation, it is more advisable to use the Neumann boundary condition. Obviously, the considerations made earlier regarding the different computational speeds updating a grid with one or the other boundary conditions do not apply to a hardware implementation.

In a grid with the Dirichlet boundary condition, the total cost is a *product* of individual costs along any path, while in a grid with the Neumann boundary condition, the total cost is a *sum* of individual costs along any path. To understand the effects of the potential decay, one can imagine a small corridor which is only one node wide and flanked by obstacle regions as shown in Figure 3.4-4. In the following considerations, each node is assumed to have four neighbours to which it is connected by four resistors. Other connectivity schemes would produce similar results.

If the Dirichlet boundary condition is applied, the potential drops at each node along the corridor by a certain factor. This factor can be calculated by formulating the problem as a linear difference equation of second order [Reimer86]:

$$\begin{aligned} a_{k+1} &= 1/4 \cdot a_{k+2} + 1/4 \cdot a_k \\ \Leftrightarrow a_{k+2} &= 4 \cdot a_{k+1} - a_k \, , \end{aligned} \qquad (3.4\text{-}20)$$

where $k = 0, 1, ..., m$ and m is the number of nodes in the corridor. The solution of this equation is $a_k = c \cdot \lambda^k$ ($\lambda \neq 0$), where c is a real constant and λ is one of the solutions of the characteristic polynomial:

$$\lambda^2 - 4\lambda + 1 = 0 \, . \qquad (3.4\text{-}21)$$

The numerical solution is $\lambda = 0.26794919$. Since Eq. (3.4-20) is homogeneous, the solution found is only correct for node activities which are infinite far away from the sink. However, the solution is a good approximation also for the inhomogeneous case considered here (see Figure 3.4-4 and [Reimer86]).

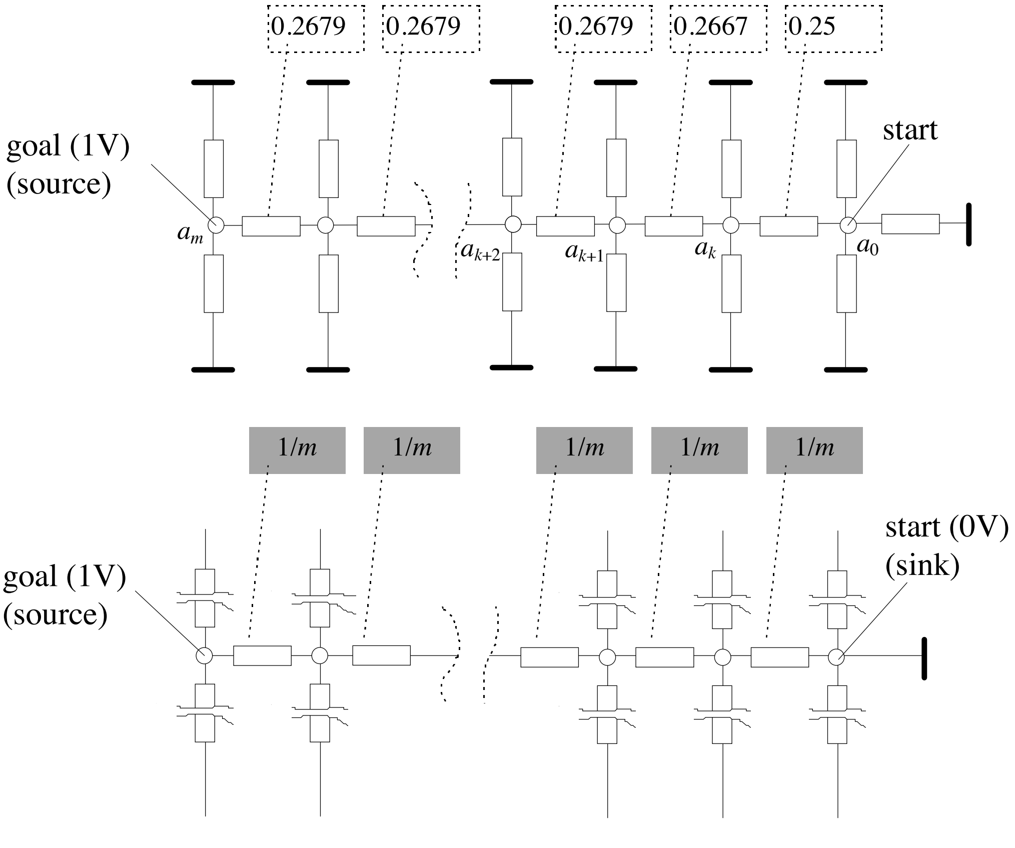


Figure 3.4-4: Corridors in resistive grids. In the case of the Dirichlet boundary condition (top), the accumulation of the costs is done using a multiplicative factor (shown in the dotted-bordered boxes). This factor converges for increasing distance from the start node. In the case of the Neumann boundary condition (bottom), the costs accumulate in an additive fashion (shown in the grey-shaded boxes). The value depends on the distance between goal and start node. Here, the goal node is m nodes apart from the start node.

The activity at any node k is $a_k \approx c \cdot (0.26794919)^k$. Assuming that the activity at the source node is 1, the activity of node k can be also expressed as follows:

$a_k \approx \frac{1}{(0.26794919)^{m-k}}$. The smallest positive number which can be processed on a computer with 64-bit floating point precision is 5×10^{-324}. (This value is based on experiments carried out in MATLAB on a Pentium90.) This means that narrow corridors, which are only one unit wide, should not be longer than approx. 564 units to assure that the gradient between two nodes at the very end of the corridor can be determined.

If the Neumann boundary condition is applied, the entire current flows along the corridor and the voltage decreases linearly. If there are m identical resistors between sink and source node, the voltage drop at each node is $\frac{source\_potential - sink\_potential}{m}$ which is a constant value (see Figure 3.4-4). In summary, longer paths can be computed in a grid due to Neumann's boundary condition than in a grid due to Dirichlet's boundary condition (see [Tarassenko91, Bugmann95]).

The exponential decay in Dirichlet grids did not cause any problems in any of the conducted experiments (see Section 3.6). An extreme case was investigated where an obstacle constellation was chosen which forced the moving robot point along a very long trajectory. (The grid's resolution was 400×400 nodes, the path between start and goal was 844 units long and the Dirichlet boundary condition was used.) Even then it was possible to read the activity distribution near the start node and to exploit the gradient to construct a path. In view of the fact that the activity distribution can be computed faster in a grid using the Dirichlet boundary condition than in one using the Neumann boundary condition, the former is to be preferred for a computer-based grid implementation to solve the kinds of planning problems described here.

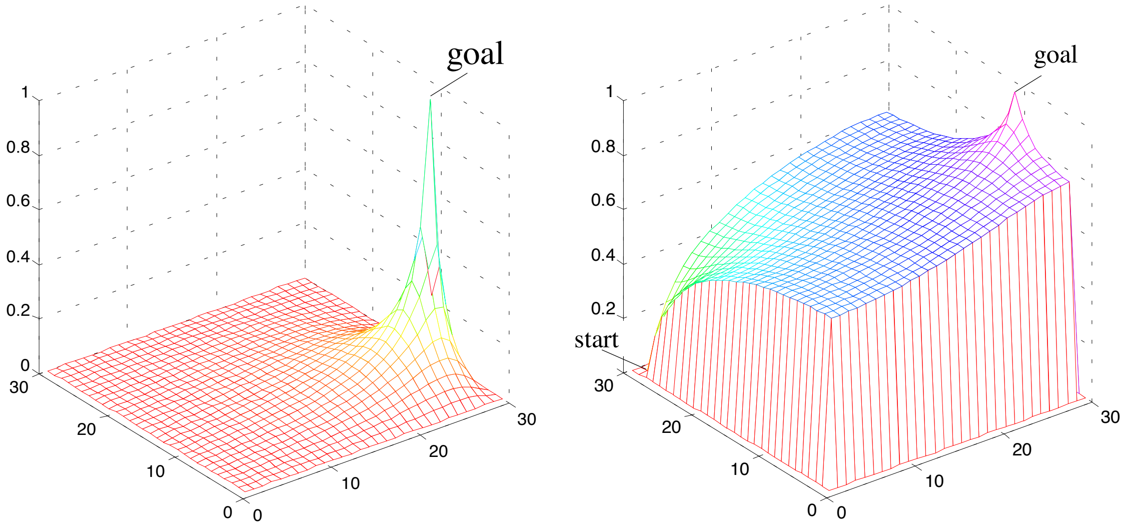


Figure 3.4-5: This figure shows the potential distribution in a resistive grid due to Dirichlet's (left) and Neumann's (right) boundary condition. Both grids were updated until convergence.

The activity distributions in 30×30-grids using Dirichlet's and Neumann's boundary conditions, respectively, are shown in Figure 3.4-5 (left) and (right). In both cases the edges of the grids are set to be obstacles. No further obstacle are introduced in this example. In both grids a positive value of 1 is applied at the source state representing the goal configuration of a planning problem. In the Neumann case a sink clamped to value 0 is introduced which represents the start configuration. It can be clearly seen that the activity distribution in

Figure 3.4-5 (left) has an exponential decay, while the decay of the activity distribution in the grid which is displayed in the right half of Figure 3.4-5 is linear over wide regions.

In both grids, a path has been planned from the start state to the goal state. For comparison the start state chosen in both grids is the one that corresponds to the state which represents the sink in the Neumann grid (see Figure 3.4-6). Beginning at the start node, the search procedure chooses at each iteration the node of the four neighbours with the highest activity. This process is continued until the goal node is found. Obviously, the results are quite different in each of the two boundary conditions. While stream lines in a Neumann grid tend to “move” along the obstacle boundaries, the Dirichlet grid provides a path which keeps distance to the obstacles (see also [Bugmann95]).

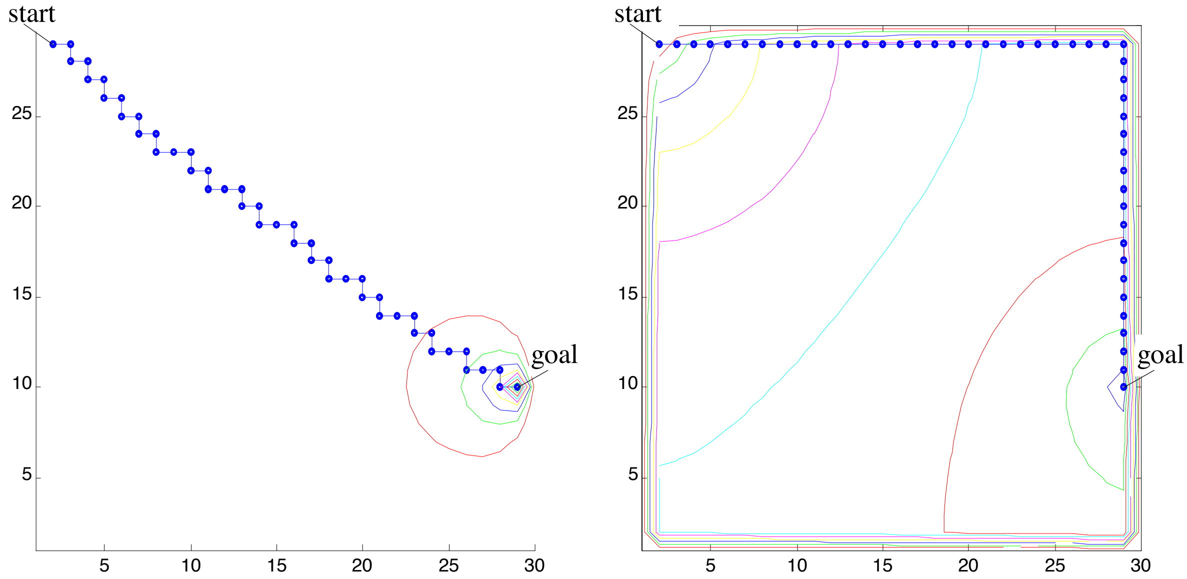


Figure 3.4-6: (left) Equi-potential lines in a grid due to the Dirichlet boundary condition. (right) Equi-potential lines in a grid due to the Neumann boundary condition. Each node in both grids has been iterated until convergence.

Both paths in Figure 3.4-6 have the same Manhattan-distance length and are the shortest ones in this discretised world (see also Section 3.7.3). Obviously, many shortest paths of the same length can be found in this example. The underlying idea is that the gradient between two nodes represents the flow of current. Following this flow of current or stream line leads inevitably to the source or goal. It is important to note that the found path is only an approximation of the actual current flow, since the gradient of the potential is equal to the current flow only in a continuous and homogeneous medium (see Section 3.4.3). At a later point in this chapter, an example will be discussed which shows that in non-homogeneous grids the calculation of the potential distribution only is not sufficient to calculate a feasible path (see Section 3.5.4).

3.4.5 Convergence Criterion for the Neuro-resistive Grid

The question remains whether the neuro-resistive grid emulated on a digital computer will always converge to one unique solution. Local extrema might occur while iterating the grid. Using an approach commonly applied to Hopfield neural networks, it will be shown that the grid always converges to one unique activity distribution which only depends on the obstacle distribution and the goal position. It will be also shown which steepness the transfer function *f* of the grid nodes is allowed to take on in order to assure convergence.

For the time-discrete Hopfield network as well as for the computer emulated resistive grid, the update algorithm to calculate the activity distribution over the lattice can be implemented as a procedure based on the relaxation method which allows the solution of a set of nonlinear equations in an iterative fashion [Connolly90, Bronshtein85, Althoefer95e, Bugmann95]:

$$\mathbf{v}^{(k+1)} = f(\mathbf{w} \cdot \mathbf{v}^{(k)} + \mathbf{V}). \qquad (3.4\text{-}22)$$

If function *f* is set to be linear with gain 1, Eq. (3.4-22) goes over into Eq. (3.4-11).

Global stability of a system which is defined by differential or difference equations can be proven using the Lyapunov function. The Lyapunov function produces a scalar which describes the behaviour of the whole system at any time. Global stability means that the system converges to a stable equilibrium. Hopfield showed that the stable states of the system described by Eq. (3.4-22) are the local minima of the following Lyapunov function [Cichocki94 and references therein]:

$$L(v) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} v_j v_i - \sum_{i=1}^{N} v_i V_i + \sum_{i=1}^{N}\int_0^{v_i} f^{-1}(x)dx. \qquad (3.4\text{-}23)$$

To see whether the Lyapunov function has a local minimum, the change of the Lyapunov function ΔL has to be examined. It can be shown that for a certain relation between the eigenvalues of the interconnection matrix and the slope of the activation function, ΔL is always negative and therefore the Lyapunov function has at least one local minimum:

$$\frac{1}{\lambda_{min}} \textit{ has to be greater than } \max(f'(x)). \qquad (3.4\text{-}24)$$

where λ_{min} is the most negative eigenvalue of interconnection matrix $\mathbf{w}$, and $f'(x)$ is the slope of the activation function. A detailed description of this proof can be found in Appendix B.

The above inequation is sufficient to prove that the neuro-resistive grid will converge. Additionally, if and only if the Hessian matrix of the Lyapunov function is positive definite then the Lyapunov function itself is strictly convex and the grid converges to *one* unique solution [Glasius94]. The following inequality can be obtained (For details see Appendix B):

$$\frac{1}{\lambda_{max}} \geq \max(f'(x)) \qquad (3.4\text{-}25)$$

where λ_{max} is the most positive eigenvalue of interconnection matrix $\mathbf{w}$, and $f'(x)$ is the slope of the transfer function. Combining Eqs. (3.4-24) and (3.4-25), one obtains:

$$\frac{1}{\lambda} \geq \max(f'(x)), \quad \lambda = \max\{|\lambda_{min}|, \lambda_{max}\} \qquad (3.4\text{-}26)$$

If Eq. (3.4-26) is satisfied, the Lyapunov function is strictly convex and the resistive grid will always evolve into a unique final solution which only depends on the shape of the obstacle regions and the location of the goal state. This solution has only one global extremum.

If the linear saturating activation function as given in Eq. (3.4-2) is used, the first derivative $f'(x)$ is the constant k and the reciprocal of the most negative eigenvalue of matrix **w** has to be greater than this constant k [Glasius94]. For the activation function generally used in Hopfield networks (tanh, linear, signum, sigmoidal) an estimation can be made for $(f^{-1})'(\xi)$ which ensures that the Lyapunov function has a local minimum. If $f(x)$ is a non-linear function which is increasing monotonously until saturation, $1/\lambda_{min}$ has to be greater than the maximum slope of $f(x)$. If the activation function is the tanh-activation function as given in Eq. (3.4-3), it follows that $f'(x) = \frac{k}{\cosh^2(kx)}$ which has its maximum at $f'_{\max}(x_0 = 0) = k$ (like in the linear case).

After choosing a value for k, the weight values of matrix **w** have to be adjusted in such a way that all the eigenvalues of matrix **w** satisfy Eq. (3.4-26). The number of non-zero elements per row or column in matrix **w** is equal to the number of non-obstacle neighbours any node is connected to. The maximum of this number is obviously given by the chosen connectivity scheme which defines how many neighbours a node can have at most. It can be shown that the largest possible eigenvalue occurs in the absence of obstacles. The eigenvalue, then, is equal to the sum of weights in a row or column of the interconnection matrix **w** (see [Bronshtein85, Glasisus94 and references therein]).

If, for example, k is chosen to be 1 and the connectivity scheme in a 2-dimensional grid connects node i to the four nearest neighbours (left and right, above and below), the sum of weights in a row or column must be smaller or equal $k = 1$ to satisfy Eq. (3.4-26). In the limits of Eq. (3.4-26), the weights can be set to ¼ (see [Bronshtein85]). In other words, it has to be assured that the sum of all non-zero elements in each row or column does not exceed the value 1. Thus, weight values for the connections in the resistive grid chosen according to Eq. (3.4-26) (as well as smaller positive values) assure that the grid converges to one unique solution.

3.5 Enhanced Activity Propagation

This section presents a novel update algorithm for the neuro-resistive grid emulated on digital computers [Althoefer95e]. The proposed update algorithm propagates the nodes' activities to and fro thereby generating within only a few sweeps a well-spread "potential" distribution suitable to perform a gradient search for path planning in configuration space. In accordance to the particular behaviour of this algorithm, it is named *To&Fro algorithm*. Applied to the specific problem of manipulator path planning in configuration space, this algorithm turns out to be particularly fast.

3.5.1 Methodology

In principle, there are many different ways to solve the linear equation system of the resistive grid. Analytical solutions are only feasible in a space where the boundaries of the forbidden areas can be modelled by simple mathematical functions (as for example done in [Tarassenko91, Wolff68]). Since the boundaries of the configuration spaces discussed here have complicated shapes (see for example Figure 3.6-1 and Figure 3.6-2), analytical solutions are not considered. Numerical methods include direct methods as well as iterative methods. Direct methods are usually not feasible because they are very memory expensive [Reimer86] and, thus, not applicable to grids with a high resolution. Iterative methods can cope with big grid sizes and provide “good” solutions after only a few iterations (see following sections). Iterative methods are said to be fast especially when the number of iterations is much smaller than the number of nodes (see [Reimer86]).

The update rule given in Eq. (3.4-11) allows the calculation of the potential or activity distribution in a computer emulated grid or neural network. So far, the sequence in which the nodes shall be updated in order to provide quickly a satisfactory activity distribution has not been discussed in detail. In view of the path planning problem under study, a satisfactory distribution of activity is one which allows performance of a gradient ascent from a start state to a goal state while avoiding obstacle regions. In this chapter, there is more interest in finding a practical solution to the path planning problem in as short a time as possible than in finding the activity distribution which most accurately resembles the potential distribution in a physical resistive grid. In other words, the emphasis here is rather on finding a collision-free path than on describing the effects of a physical phenomenon.

An update sequence will be proposed which is especially adapted to the problem of path planning for robotic manipulators. This sequence makes use of the fact that the set of possible paths from a start to a goal configuration is in many applications of a relatively simple nature. Labyrinth-like problems which are for example possible in mobile robot applications do not normally occur (see [Millan92, Bugmann95]).

```
INITIALISATION
        all obstacle nodes are clamped to zero
        goal node is clamped to 1
        initial condition: all “free” nodes are set to zero

LOOP until node activities have settled
  { FOR each of the 4 corners of the resistive grid DO
     { FOR all columns from the corner to opposite corner in horizontal direction DO
        { FOR all nodes in the column from the side of the corner to opposite side DO
           { update node as described in Eq. (3.4-11)
           }
        }
     check whether start state has an activity greater than 0 and BREAK
     }
  }
```

Figure 3.5-1: Pseudo-code of the To&Fro algorithm. Once the start state has an activity greater than zero, the activity distribution has spread far enough to construct a path towards the goal state. If after the settling of the activities the start node’s activity still remains on zero, no path exists at the chosen grid resolution.

The C-Space of a manipulator is described by its variable parameters $\theta_1, \theta_2, ..., \theta_n$. The C-Space is represented by a discrete grid where each *grid node* is connected via resistors to the neighbours. Two different connectivity schemes have been investigated: firstly, each node has vertical and horizontal neighbours only, secondly, in addition to the vertical and horizontal neighbours each node has diagonal neighbours.

Theoretically, after an infinite number of updates, each node's activity would be equal to the average activity of the neighbours it is connected to. It has been reported that this state can be reached after N^2 updates (see [Cichocki94]). A grid with this activity distribution approximates the potential distribution in a physical resistive grid after reaching equilibrium. However, it appears that for manipulator path planning it is not necessary to find the final solution and, the number of updates per node necessary to find a collision-free path is surprisingly small.

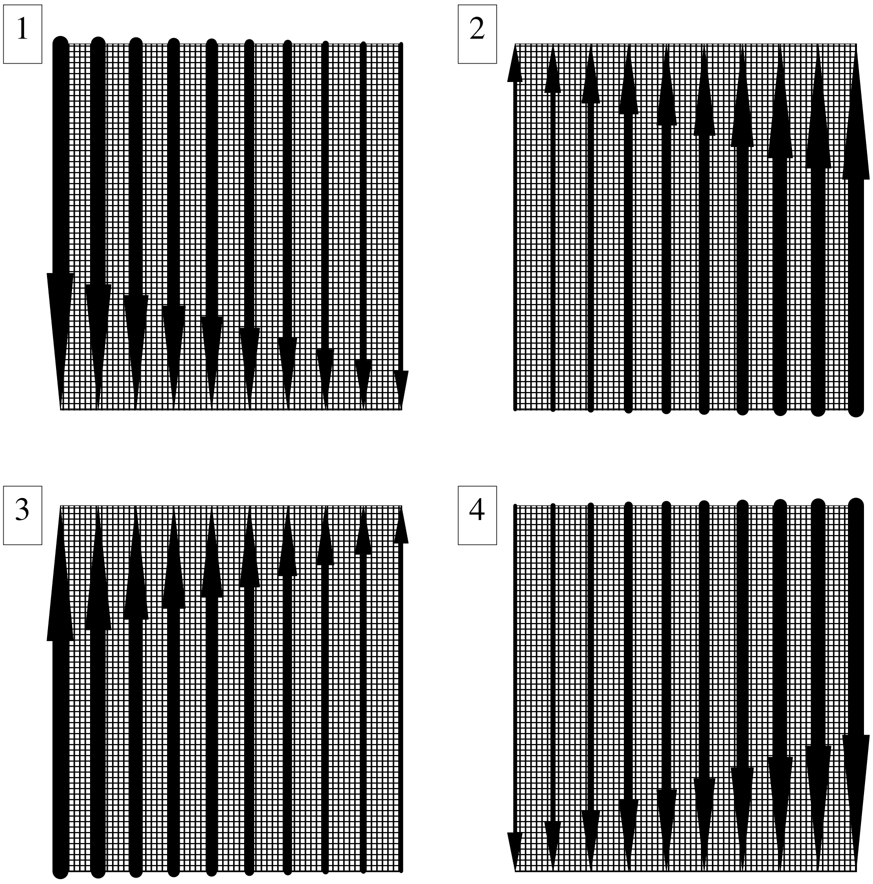


Figure 3.5-2: Depiction of the update procedure according to the To&Fro algorithm in a two-dimensional grid. The four different sweep procedures are repeatedly applied to the grid in the order 1-2-3-4. Each individual sweep procedure begins updating in a different corner of the grid. This is denoted by the thickest arrow. Subsequent updates are symbolised by arrows which shrinking width. Nodes are always updated in vertical direction from the arrows end to its tip.

As shown in Chapter 2, for a two-link manipulator, an obstacle point colliding with link l_2 transforms into an *s-shaped* obstacle area in C-Space (see Figure 3.6-5). Any of those C-Space obstacles evolves around the centre line where $\theta_2 = 0$. It can be observed that the expansion in vertical direction is much greater than the expansion in horizontal direction. This argument is also valid for obstacles which cause a collision with the first link or both links. Furthermore, a similar comment applies for more complex obstacles, because any workspace obstacle can be represented by a set of obstacle points (see Chapter 2 and [Newman91, Althoefer95e]).

The main key to the proposed To&Fro algorithm is that the evolution of the activity distribution is strongly influenced by the order in which the nodes are updated. Due to the predominance of the C-Space obstacle expansion in the vertical direction, it is most effective to update the nodes along the same direction propagating the potential distribution to and fro (see also [Hockney88]). This is outlined in Figure 3.5-1 for a two-dimensional grid using Dirichlet's boundary condition. This figure depicts the To&Fro algorithm in a pseudo-code.

The update procedure is visualised in Figure 3.5-2. It can be seen that for each of the four sweeps the nodes are always updated in the same direction - either from top to bottom or bottom to top. A complete update cycle is finished after four sweeps. Due to the fact that these sweep procedures change in direction, the activity distribution "bounces" to and fro. Alternative ways of updating the grid (change of direction after a column is completed, vertical and horizontal update directions) have been investigated. Those update methods did not provide satisfactory activity distributions in a short time.

A similar update rule (as depicted in Figure 3.5-2) can be used for a grid due to Neumann's boundary condition. In this case goal and start are clamped to the activity values 1 and 0, respectively. Obstacles nodes are insulated from nodes representing free space, thus, in the calculation only weights are considered which connect the actual node to unclamped nodes (see Section 3.4.4). A slight variation to the above given algorithm (Figure 3.5-1) is necessary to check whether the activity distribution has spread far enough. Since the start node is clamped to 0 and therefore will never change its activity, the neighbourhood of the start node has to be inspected for a rise in activity.

If no path can be found, because none exists at the chosen grid resolution, a different way has to be applied to terminate the update process. One possible way is to terminate the update process once the grid's activities have converged. As shown later, only a small number of updates is necessary to generate an activity distribution to find a path, if such path exists. Extensive experiments showed that if after a few updates per node, no path can be found, it can be assumed that no path exists.

3.5.2 Higher Dimensions

Although the focus in this part of the chapter is on path planning for a two-link manipulator, the presented update method can be expanded to higher dimensions. Similar considerations regarding a fast spreading of the activity distribution have to be made.

For the experiments with the three-link manipulator, single sweeps of the aforementioned To&Fro algorithm have been applied to the C-space of links l_1 and l_2 while the angle of link l_3 is kept fixed. This process is repeated for discrete values of the joint of link l_3 changing its angle values from the most negative angle value to the most positive one and vice versa (see Figure 3.5-3). Thus, the process of spreading the activity distribution in a two-dimensional space is elevated and lowered repeatedly along the axis of the third dimension to accomplish the activity distribution in a three dimensional space. When one complete update cycle has finished, each node has been updated eight times. The process stops, once a path has been found or after a certain number of iterations have been carried out without success. Experimental studies carried out in the following sections, showed that usually a path could be constructed before a complete update cycle was finished. (A complete cycle consists of the eight sweeps as shown in Figure 3.5-3.)

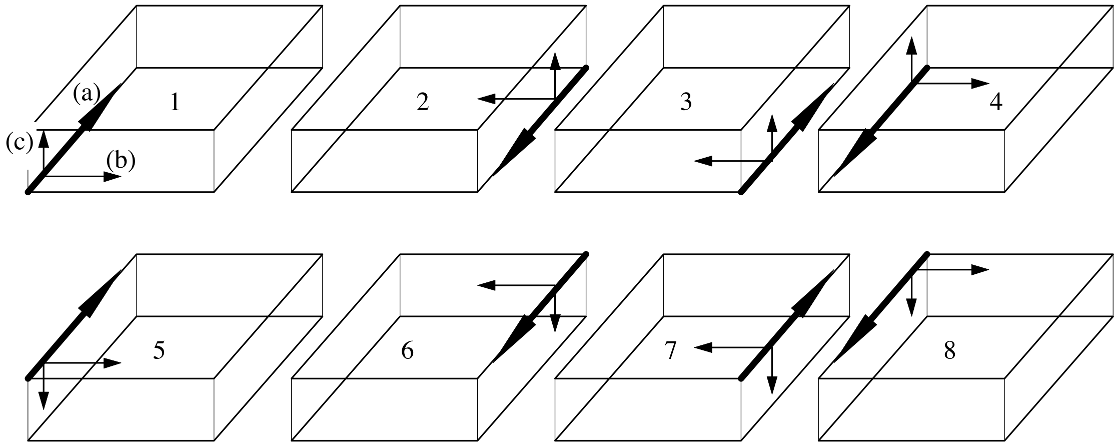


Figure 3.5-3: Update order in a three-dimensional space. The bold arrows (a) indicate the update process along the first column. The slim horizontal arrows (b) indicate how the column-wise update process proceeds along the two-dimensional space. The vertical arrows (c) depict the continuation of the update process in the third dimension. Numbers indicate the sweep order.

The update method can be also extended to be applied to three-dimensional manipulators. For example, the three-dimensional configuration space of a three-link manipulator whose shoulder and elbow link pivot around parallel axes and which are perpendicular to the axis of the waist joint can be built by a collection of two-dimensional configuration spaces [Newman91, Branicky90]. In case the waist joint is a revolute one (for example PUMA series and MA 2000), the two-dimensional configuration space of shoulder and elbow link is rotated around its *y*-axis. In case the waist joint is a prismatic one (for example SCARA robots) the two-dimensional configuration space of shoulder and elbow link is piled up -one on top of the other- along a *z*-axis which is perpendicular to the *x*-axis and *y*-axis of the two-dimensional C-spaces (see Figure 3.6-17). Obviously, the constructed three-dimensional configuration space has great similarity to the one of the planar three-link arm. Hence, the update method can be also applied to three-dimensional arms.

3.5.3 Global Extremum and Collision-free Path

The following description shows that the activity distribution due to an interrupted update procedure does not produce local extrema (here: local maxima[3]). The description will focus on a grid employing Dirichlet's boundary condition. (A detailed investigation of a grid due to Neumann's boundary condition has not been carried out, but the conducted experiments show the same overall behaviour.) The goal node is constantly hold at activity 1, while obstacle nodes are clamped at all times to activity 0. Only unclamped nodes can change their activity during the update process. The update algorithm used is the To&Fro algorithm. For simplicity, only grids are considered where nodes are connected via resistors to their two horizontal as well as their two vertical neighbours.

For the investigation of the events in the grid during updating, two aspects are important: the initial condition, and the update rule. Initially, only the goal node is set to an

[3] In all conducted experiments, the goal is set to a positive value which is desired to be the one and only maximum. If a local maximum would occur, a path to the goal might be missed. However, multiple minima may appear but do not prevent the path searching algorithm from finding a path to a global maximum.

activity of value one, while all other nodes are set to zero. The update rule given in Eq. (3.4-11) is applied to every node of the grid. This rule adds up the activities of neighbouring nodes and divides this sum by the number of neighbours. The result which represents the average of the neighbour's activities becomes the new activity of the node being momentarily updated.

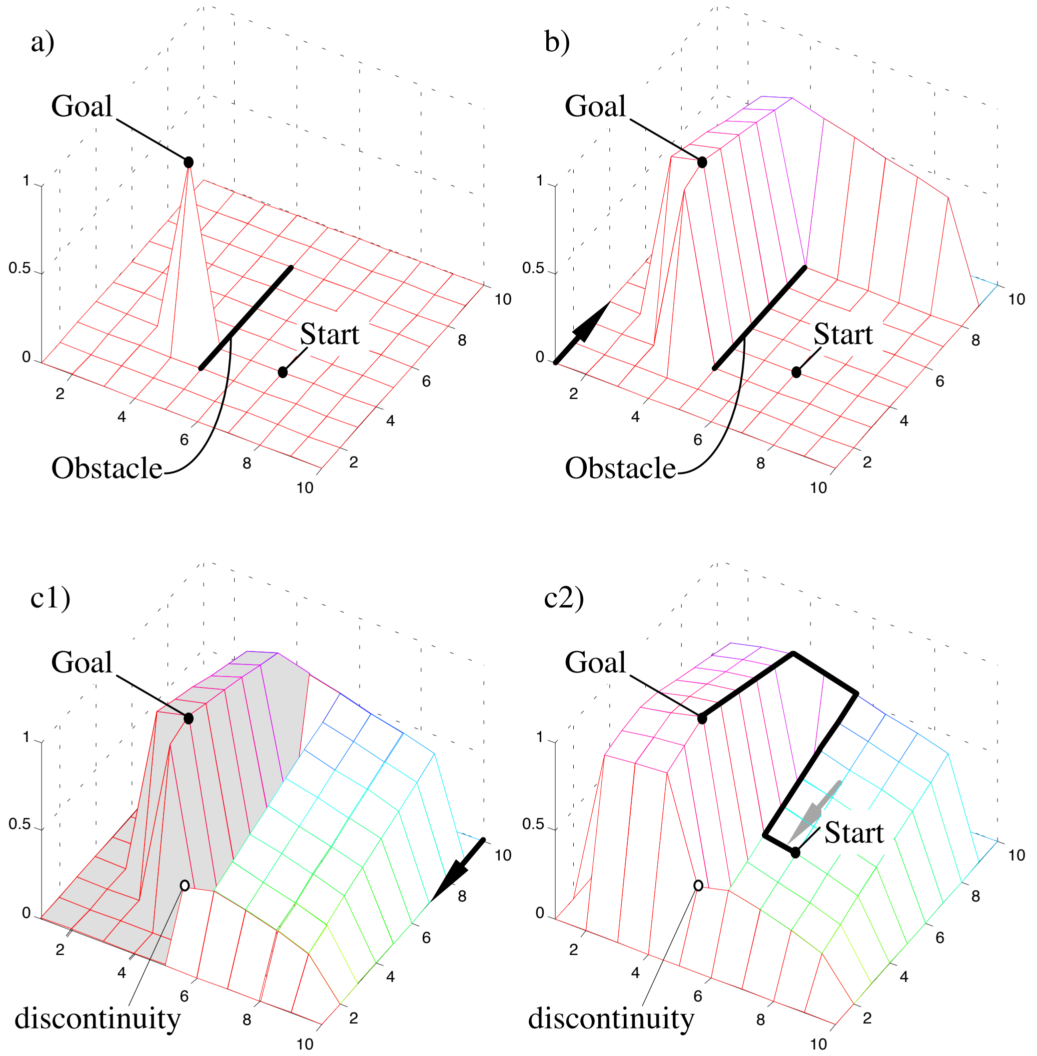
When the update algorithm sweeps the first time over the grid, nodes which are visited before the neighbourhood of the goal node is reached do not change their activity, since the average of zero remains zero. Once the nodes in the neighbourhood of the goal node are reached, these nodes will experience a rise in activity. The activity of unclamped nodes which are visited thereafter until the current update cycle is finished will also rise. Only those unclamped nodes which are in the "shadow" of obstacle regions will not change their activities. When the To&Fro algorithm is employed, the update direction is reversed at the end of each cycle (see Figure 3.5-4) and the activity of the nodes is propagated against the previous direction and so on. This local averaging process can be seen as a spatial low pass filtering which smoothes the contour of the activity distribution which consists initially of a single peak (goal) in an otherwise completely flat region (initial condition) (see also Figure 3.5-4 (a)).

The To&Fro algorithm changes the activity of a node into an activity which is average among the activities of the neighbours. This means that whenever node *i* is updated its activity is smaller than the maximum neighbouring activity and greater than the minimum neighbouring activity, or, if all neighbours have zero activity, the activity of node *i* remains also zero. Clearly, the activity of node *i* never exceeds or falls short of neighbouring activities. Since all unclamped nodes have initially zero activity and the goal node has a positive activity the outcome of node updates in the first sweep will cause an activity increase in those nodes which are updated after the goal node has been updated. In subsequent sweeps, the averaging process will increase the activity of unclamped nodes. At no time a decrease in activity can occur. During the repeated update process, the node activities rise until convergence is reached. At this point, each unclamped node's activity is the average of neighbouring activities and no further change takes place.

Before convergence, the activity of an unclamped node *i* is the average of its neighbours' activities only in that moment when the node is updated. Since the update process proceeds, neighbours of node *i* will be updated and change their activity after node *i* has been updated. Due to these changes node *i* no longer has an activity which is average among the neighbouring activities until node *i* is updated again in the next sweep. Since any node can only experience a rise in activity, a yet not-updated node whose neighbourhood has been already updated can only develop into a local minimum (discontinuity), but not a local maximum (see also Figure 3.5-4). Thus, whenever the update process is interrupted, any of those nodes which have already experienced a rise in activity has at least one neighbour with a higher activity than its own (see also Figure 3.5-4). A path search algorithm which follows the greatest gradient will reach the global maximum (goal) without getting stuck.

How such a discontinuity develops can be understood by examining Figure 3.5-4 c1) and c2). The activity spreads column-wise from the back to the front of the grid beginning with the column indicated by the arrow. Although the updating is carried out along a column, the activity can spread along rows because the connections to horizontal neighbours assure that the activity in an adjacent column is dragged along. When an obstacle region is reached, the horizontal spread is hindered and the spreading of the activity on the other side of the obstacle region develops independently.

The node where the discontinuity occurs is marked in Figure 3.5-4 c1) and c2). Figure 3.5-4 c1) shows the prevailing activity distribution when the second sweep reaches the obstacle region. The activity distribution produced by the first sweep is depicted by shaded areas in Figure 3.5-4 c1).



(See caption next page)

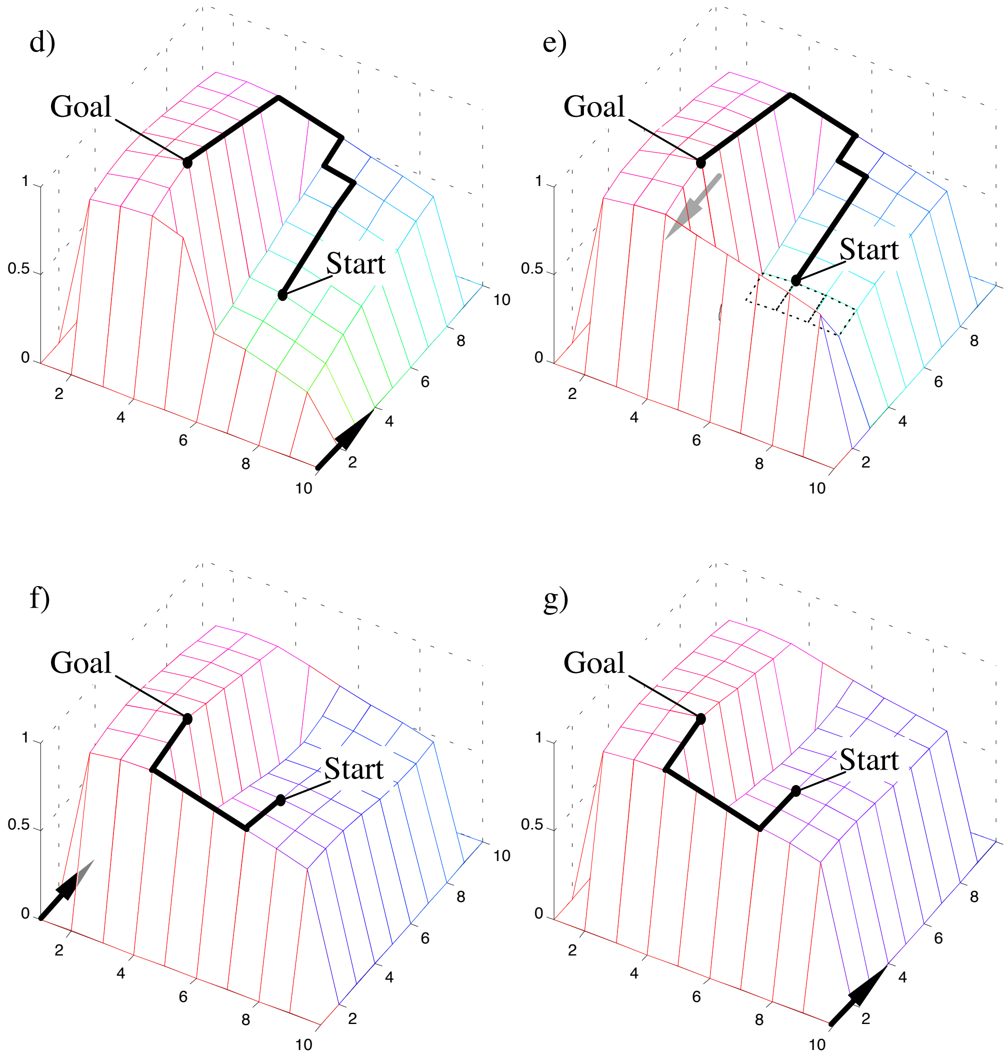


Figure 3.5-4: Activity distribution in a 10×10-grid. The used update algorithm is the To&Fro algorithm. The update direction is indicated by bold arrows. Each node is connected to four neighbours. Additional to the obstacle region depicted in a) and b), all edges are clamped to zero. The initial activity distribution is shown in a). The activity distribution is shown after b) 1 sweep, c1) and c2) 2 sweeps, d) 3 sweeps, e) 4 sweeps, f) 5 sweeps, g) 39 sweeps. Due to the discontinuity which stays up to the fourth sweep (visible in c) - e)), the found path is not the shortest one. The To&Fro algorithm would already terminate after two sweeps because the start node sensed an increase in activity. After 5 sweeps (shown in f)) the discontinuity is smoothed out and the activity distribution resembles a distribution where the maximum change in activity is virtually zero (see g)). The path shown in f) and g) has the smallest Manhattan distance. Note: the activity is depicted in a logarithmic scale. Further details see text.

The activity of the marked node is the average of the two vertical neighbours which are both obstacles (above: the lower end of the obstacle region, below: the grid edge) and the two horizontal neighbours which are not obstacles. Since in this case the update process develops column-wise from right to left, the activity of the neighbour which is on the left of the marked node is still zero (see Figure 3.5-4 c1)). The marked node's activity changes to a quarter of the right neighbour's activity, since all other neighbours have zero activity in this moment. The next column update increases the left neighbour's activity. Since the update process continues to the left, the marked node is not further updated during this round. Clearly, at the end of the second sweep the activity of the marked node is not the average of the neighbours activities. Despite the problem of discontinuity only local minima can occur - but no local maxima.

Note that sweeps three and four do not produce further discontinuities, but make the discontinuity (which was produced during the second sweep) wander along the grid until it is remedied with the fifth sweep. The comparison between Figure 3.5-4 f) and g) shows that after five sweeps only the activity distribution has more or less reached the final contour (convergence). The final contour of the grid's activity is depicted in Figure 3.5-4 g) where the change in activity became virtually zero.

In such an activity mountain, a path can be found by following the maximum gradient. Thus, beginning at any node in the grid, the neighbouring node with the highest activity among the set of neighbours indicates always the next step on a collision-free path. Since as shown in the previous paragraphs no local maxima can occur, the path will lead to the goal. As it can be seen from Figure 3.5-4, the found path is not necessarily the shortest one (see also Section 3.7.3).

The path is collision-free at all times because from any unclamped node whose activity has already risen above zero the path is constructed along a neighbour with a higher activity - never towards one with a lower activity. Thus, obstacle nodes, which are kept on zero activity, will never be crossed.

3.5.4 A Non-Topologically-Ordered Grid

In this section, a resistive grid is investigated whose nodes are connected via resistors with varying resistances. In this grid, connecting resistors or weights do not necessarily describe the Euclidean distance. Those weights define generally a cost which has to be paid in order to traverse from one node or state to another. Furthermore, nodes which are connected to each other do not need to be neighbours in the Euclidean sense. Any sequential optimisation problem can be formulated as such a grid structure (see also [Cooper81, Bellmann57]).

A non-topologically-ordered grid does not approximate a continuous medium and Laplace's equation is not necessarily satisfied. A similar argument applies to grids whose node distances are so large that the approximation in Eqs. (3.4-15), (3.4-16) and (3.4-17) cannot be carried out without producing an intolerably large error. In any case, Kirchhoff's current law is still valid and Eq. (3.4-11) can be still used to compute the activity distribution.

In such a grid the difference between the potentials of two nodes connected to each other via a resistor does not represent the current flowing. Thus, stepping from one node to the neighbour with the highest activity (as done in previous sections) does not necessarily produce the desired path. To incorporate the different costs into the search, the current through the resistors connecting the actual node to its neighbours has to be calculated prior to each transition. The largest current indicates which way to go, since the current is larger the smaller the cost or resistor is.

To describe the properties of such a grid, the example depicted in Figure 3.5-5 is discussed. In contrast to earlier investigations, nodes are connected by weights of different values. As usual, the goal node is connected to a high potential and functions as a source, while the start node representing the sink is grounded. This approach resembles a grid using Neumann's boundary condition. The grid has been updated until convergence using the update rule given in Eq. (3.4-11). Figure 3.5-5 clearly indicates that following the gradient of potentials would lead to a path with a higher cost than the one chosen along the calculated current values.

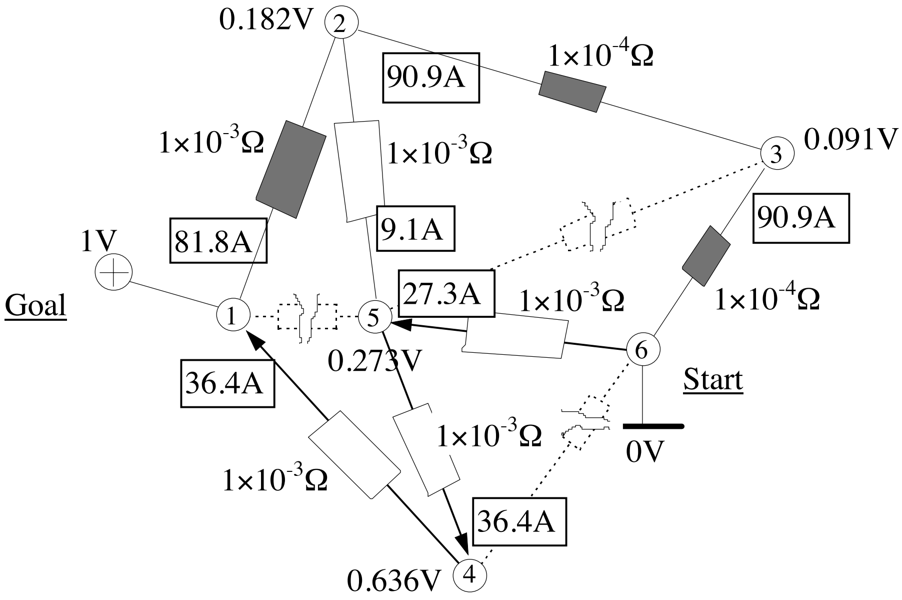


Figure 3.5-5: A non-topological ordered grid. The rectangular boxes represent resistors or costs of transition. Note: the resistance of the resistors between nodes 6 and 3 as well as between nodes 2 and 3 are ten times smaller than the resistance of the other labelled resistors. Furthermore, the resistors between nodes 1-5, 5-3 and 4-6 are cut out and these connections have therefore an infinite resistance. The path with the minimal overall cost goes along nodes 6-3-2-1, indicated by the grey resistors. This path has been found by following the maximum current flow. If the potentials had been exploited for solving this problem, the found path would wrongly lead along nodes 6-5-4-1 (indicated by arrows), which has a higher overall cost.

3.5.5 Soft Safety Margin

A variation of the resistive-grid method which calculates the flow of current through its resistors (Section 3.5.4) can be used to add a soft safety margin around obstacles. A safety margin is usually applied in path planning problems where the position of the obstacles is only known up to a certain precision.

In most applications, the obstacle regions are simply expanded in size to achieve a safety margin. In other words, obstacles are replaced by larger obstacles. Any trajectory found by a path planning strategy in such an environment keeps a minimum distance towards the original obstacles. The disadvantage of this approach is that some of the expanded obstacle regions might merge with others thereby blocking a possible passage in the original environment. In some instances, it may be useful to plan a path through such a gap. In this case, the robot could for example move fast in areas outside the safety margin, and slow down while traversing the gap. The movement through the gap could be supported by external sensors.

The method suggested here assures, on the one hand, that the found trajectory keeps a minimum distance to those obstacle regions which are surrounded by obstacle-free areas. On the other hand, in narrow but passable gaps between two obstacle regions where the safety margins of two obstacles merge the planner is able to penetrate the area of the safety margin and traverse the gap. This is achieved by using a resistive grid with resistors whose resistance near to obstacles is higher than those in the obstacle-free areas. The result is that most of the current flows through the low-resistance resistors for most parts of the grid. However, gaps between obstacles are still passable, as the current can still flow through the high-resistance resistors.

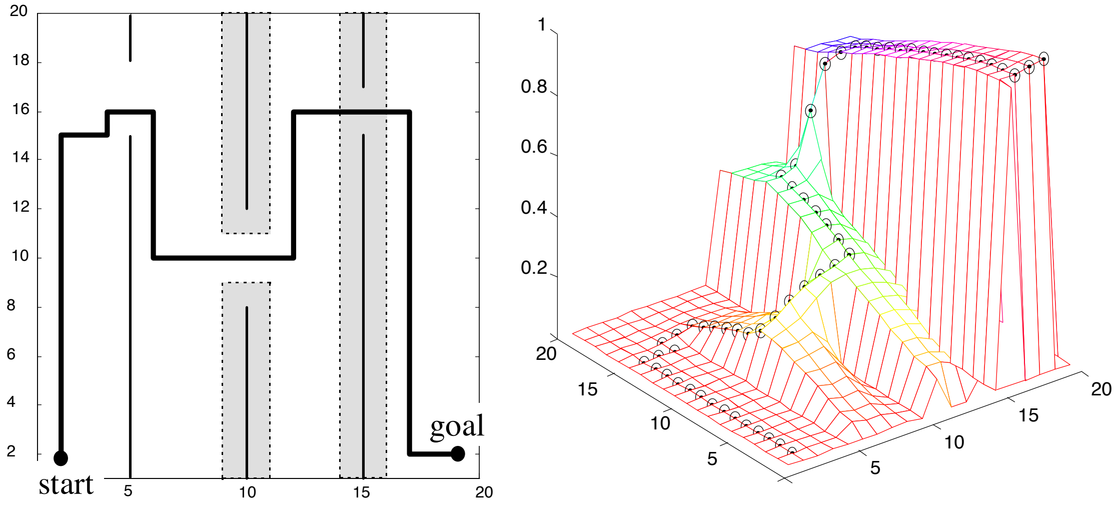


Figure 3.5-6: The soft safety margin in a resistive grid. The safety margins (high resistor values) are depicted by the grey areas in the workspace on the left. The grid size is 20×20. The To&Fro algorithm produced after 7 sweeps an activity distribution which could be exploited for constructing the shown path. The ascent towards the goal was undertaken by following the maximum current. The right subfigure shows the final activity distribution. Note that the *z*-axis has a logarithmic scale. For further explanations see text.

Figure 3.5-6 (left) depicts an example for a point-sized robot in an environment cluttered with obstacles. The two line obstacles on the left have no safety margin. The obstacles in the centre and on the left of the workspace are surrounded by safety margins represented by resistors with a high resistance (grey regions). The safety margin is one unit wide. In this experiment the ratio between resistor values inside the safety margins and those in the obstacle-free areas was set to 25:1. Further experiments carried out showed that other ratios lead to the same result.

As one can see from Figure 3.5-6 (left), the found trajectory comes close to the lower left obstacle which has no safety margin, but the trajectory stays clearly outside of the safety margins of the other two obstacles, and successfully traverses the gap between the two line obstacles on the right of the workspace.

3.6 Experiments

3.6.1 Real-World Experiments with the MA 2000 Manipulator

This section describes the real-world experiments carried out. The manipulator in use is the MA 2000 which is an experimental manipulator with three main joints: waist, shoulder and elbow. Further details on this manipulator are given in Appendix A-1. This section focuses on planning a path from a given start to a given goal configuration for the shoulder and elbow links of the MA 2000 manipulator. This is a planar planning problem as the axis of these two joints are parallel. The elbow link is extended by a further link, named finger. The joint of the finger is fixed to a constant configuration of 0 for the following considerations.

All experiments had been carried out on an IBM-PC-compatible with a Pentium90. The software package MATLAB has been used to speed up the production of a representative graphical display of the activity distribution in the grid. All the programs have been encoded in the programming language which is part of MATLAB. Since this MATLAB version (Version 4.0) does not compile but only interprets the program code, the running time of the programs is very long, a fact that does not allow a meaningful comparison to algorithms generated by other researchers.

For this reason the core of the update routine has been converted into a program written in the programming language C. The core of the update routine is outlined in Section 3.5. The compiled C programs have been also run on a Pentium90. This has been done to achieve a better idea of the possible speed of the proposed update algorithm. Of course a routine written in Assembler would provide further improvement in terms of speed. This applies especially to the resistive grid using Dirichlet's boundary condition because a division by four or eight can be accomplished by fast shift operations.

In Table 3.6-1 the running times for different grid sizes are depicted. One sweep represents one of the four update cycles according to the To&Fro algorithm (see Figure 3.5-2). During one sweep every node is updated once. The running times of the corresponding MATLAB program are shown for comparison purposes. Table 3.6-1 shows that the C-program is 180 to 200 times faster than the MATLAB program[4]. Table 3.6-1 also reveals that the running times are approximately proportional to the number of nodes in the investigated grids (see also Figure 3.6-4).

grid size	number of nodes	time per sweep in seconds		time in seconds
		C-program 8 neighbours	MATLAB 8 neighbours	MATLAB gradient search
50×50	2500	0.009	1.8	0.16
75×75	5625	0.021	4.26	0.17
100×100	10000	0.036	7.47	0.22
125×125	15625	0.056	12.06	0.33
150×150	22500	0.088	17.96	0.44
180×180	32400	0.126	26.67	0.55
300×300	90000	0.375	71.69	1.87

Table 3.6-1: Comparison of running times. Obviously, the times are proportional to the number of nodes. The update algorithm encoded in C is 180-200 times faster than the MATLAB version. The time spent on searching the gradient of the activity distribution to find a path is negligibly small.

Columns 3 and 4 in Table 3.6-1 show the times needed to built the activity distribution in the grid of varying sizes. It has been found that the time necessary to carry out the gradient

[4] Informative material published by MATLAB states that their newly developed compiler speeds up programs by a factor of 100 to 200 [Matlab96].

search to construct the path after the To&Fro algorithm has finished is only a fraction of the update time (see last column of Table 3.6-1).

For the experiments described here, a path was planned for the two-link combination of the MA 2000's shoulder and elbow links in a planar workspace with three point-sized obstacles. The origin of the workspace is at the centre of the axis of the shoulder joint. The location of the three point obstacles are, as follows (see also Figure 3.6-1 and Figure 3.6-2): A: $(x_A, y_A) = (176.9722, 212.7059)$ *mm*, B: $(x_B, y_B) = (108.8051, 358.9412)$ *mm*, C: $(x_C, y_C) = (-255.6265, 116.3235)$ *mm*.

The C-space patterns corresponding to these obstacle points were calculated using the method described in Chapter 2. For an overview of the C-space patterns of the MA 2000 (see Appendices A-2 and A-3). The C-space patterns are loaded into a map which represents the configuration space in discrete form. In the following experiments, the map as well as the resistive grid are quadratic arrays of varying sizes.

Any node in the map representing a forbidden configuration is set to the value -1; any node representing C-free space is set to 0. The goal configuration is set to 1. Each node of the map is connected to a node in the resistive grid at the same location (see Figure 3.4-1). The resulting configuration space is depicted in Figure 3.6-1 and Figure 3.6-2. In addition to the three C-space obstacles, the C-space boundary which describes the possible workspace of the manipulator has been set to the obstacle value -1. The workspace of the manipulator has been determined by moving the manipulator in an obstacle-free environment. Configurations which represent collisions with the base of the manipulator, the workbench, and between links represent the C-space boundary (see Figure 3.6-1, Figure 3.6-2 and Chapter 2).

In Figure 3.6-1 a resistive grid with a size of 300×300 nodes is shown. The To&Fro algorithm has generated an activity distribution indicated by the dotted equi-potential lines. In these experiments each node is connected to eight neighbours. The figure also depicts the found path from the start configuration $(\theta_{1\text{start}}, \theta_{2\text{start}}) = (0.5792, 0.064)\cdot\pi$ to the goal configuration $(\theta_{1\text{goal}}, \theta_{2\text{goal}}) = (-0.111, -0.1796)\cdot\pi$. The s-shaped C-obstacle regions are successfully bypassed. Figure 3.6-2 shows a 50×50 grid which is applied to the same obstacle scenario. In contrast to Figure 3.6-1, goal and start configuration are swapped.

Figure 3.6-1 and Figure 3.6-2 show that the number of sweeps does not only depend on the obstacle constellation but also on the location of goal node in the grid. The To&Fro algorithm sweeps over the configuration space regardless of the position of start and goal node. The earlier the goal node, which represents the source, is "reached" by the update algorithm, the faster the distribution spreads over the grid. An improved version of the To&Fro algorithm could take the location of the goal node into account and begin updating at this location.

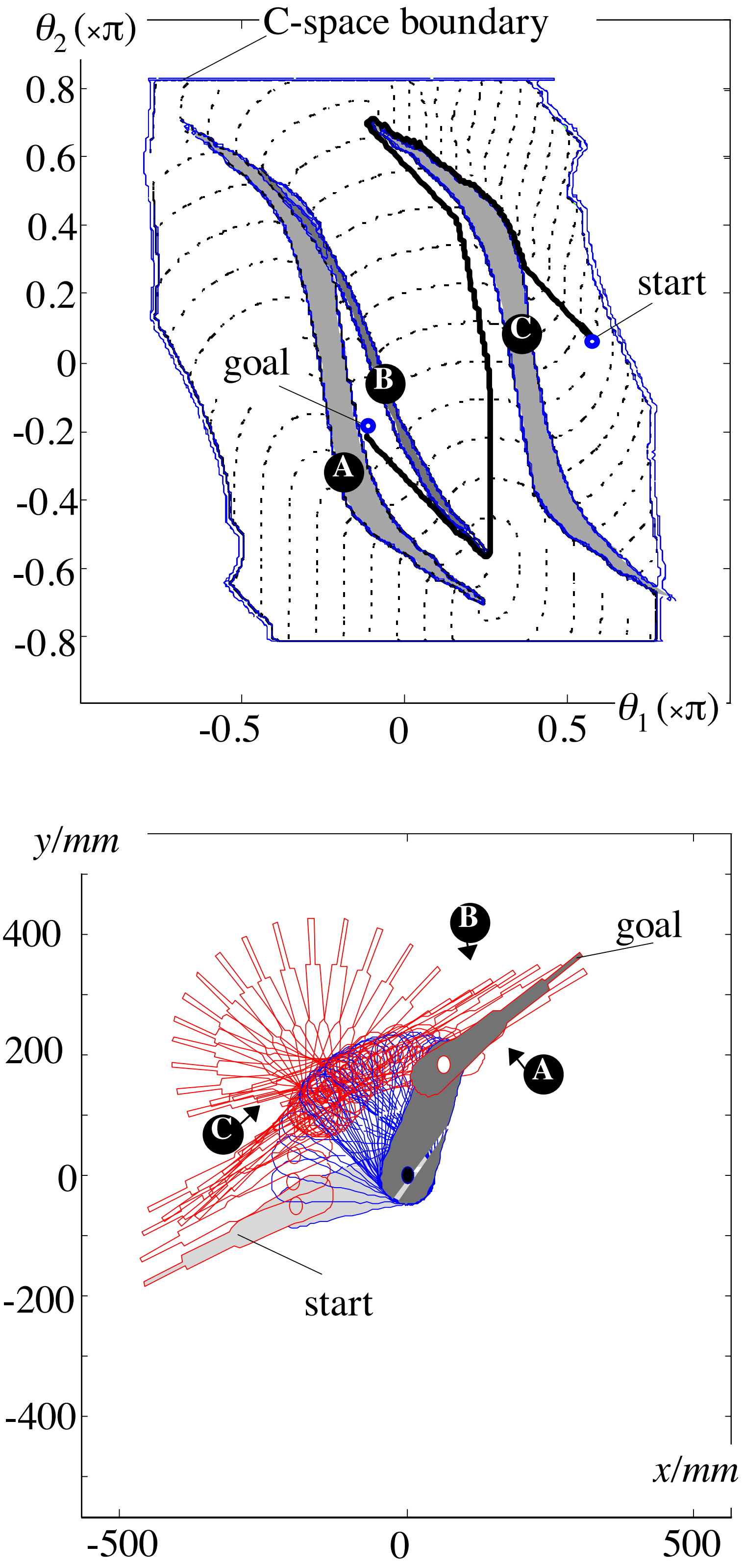


Figure 3.6-1: (top) Trajectory found by the To&Fro algorithm in a 300×300 grid. The resolution per joint is 1.2°. To calculate the activity distribution the routine needed seven sweeps which took 502.45 sec. under MATLAB. (bottom) Each tenth step along the trajectory of the MA 2000 model has been depicted. The actual locations of the obstacles are at the endpoints of the arrows.

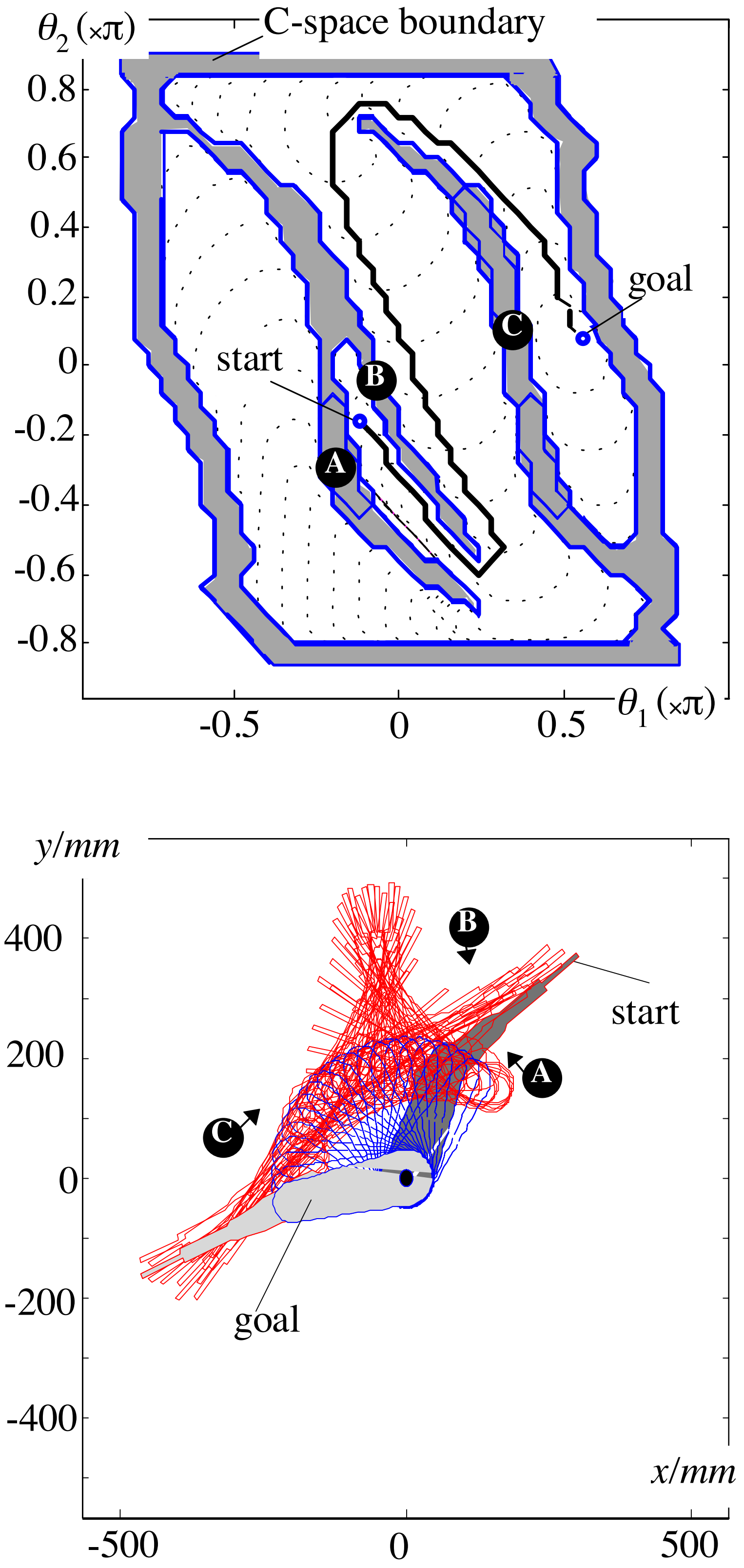


Figure 3.6-2: (top) Trajectory found by the To&Fro algorithm in a 50×50 grid. To calculate the activity distribution the routine needed five sweeps which took 9 sec. under MATLAB. In contrast to Figure 3.6-1 goal and start a swapped. (bottom) Every step along the trajectory of the MA 2000 model has been depicted. The actual locations of the obstacles are at the endpoints of the arrows.

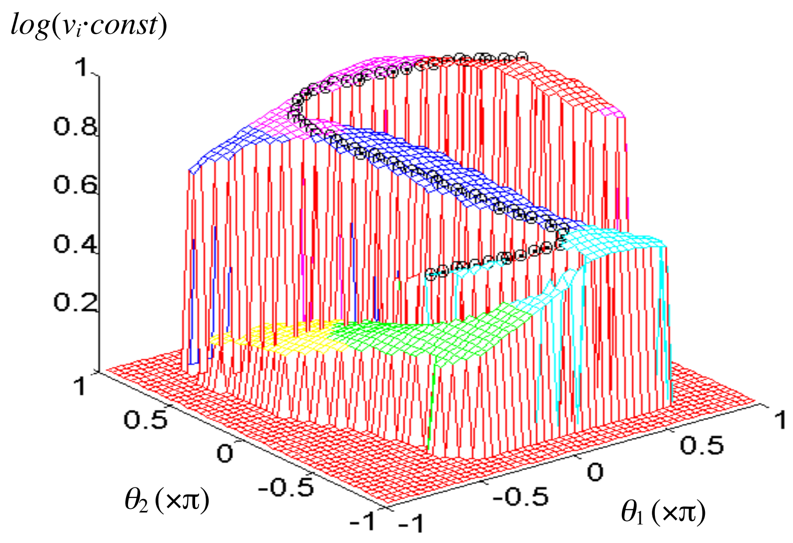


Figure 3.6-3: Three-dimensional depiction of the activity distribution in the 50×50-grid (see Figure 3.6-2).

Since the goal and start location are swapped in the two experiments above, the number of sweeps is seven (Figure 3.6-1) and five (Figure 3.6-2), respectively. The execution time under MATLAB was 502.45 seconds in the 300×300-nodes grid and 9 seconds in the 50×50-nodes grid. In view of Table 3.6-1, this means the path could be calculated in approximately 2.6 seconds (300×300 nodes) and 0.04375 seconds (50×50 nodes) executing a compiled C-program on a Pentium90. The sequence of configurations which comprise the found path are applied to a model of the manipulator as shown in Figure 3.6-1 (bottom) and Figure 3.6-2 (bottom).

grid size	no. of nodes	8 neighbours				4 neighbours			
		start-goal		goal-start		start-goal		goal-start	
		swps	total time	swps	total time	swps	total time	swps	total time
50×50	2500	5	9	7	12.6	6	10.755	8	14.34
75×75	5625	5	21.3	7	29.82	6	24.593	8	32.79
100×100	10000	5	37.35	7	52.29	6	43.83	8	58.44
125×125	15625	5	60.3	7	84.42	6	69	8	92
150×150	22500	5	89.8	7	125.72	6	98.7	8	131.6
180×180	32400	5	133.34	7	186.676	6	143.6	8	191.2
300×300	90000	5	358.49	7	501.886	6	406.793	8	542.39

Table 3.6-2: Execution times for grids of different sizes under MATLAB. The "total time" includes the path finding time. In the "start-goal" columns, the start- and goal-configuration are as follows: $(\theta_{1start}, \theta_{2start}) = (-0.111, -0.1796){\cdot}\pi$; $(\theta_{1goal}, \theta_{2goal}) = (0.5792, 0.064){\cdot}\pi$. In the "goal-start" columns, those values are swapped.

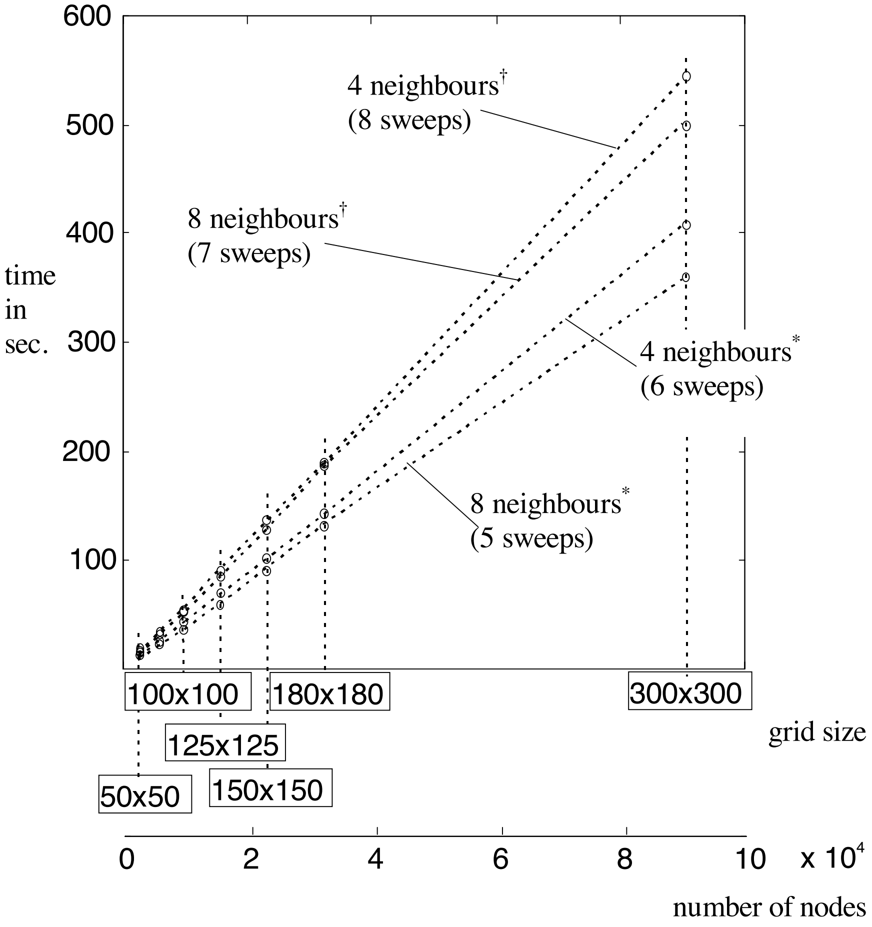


Figure 3.6-4: Performance of the To&Fro algorithm. Execution time vs. number of nodes. The execution time increases linearly with time. The execution time depends also on the position of goal and start node (* $(\theta_{1\text{start}}, \theta_{2\text{start}}) =$ (-0.111, -0.1796)·π; $(\theta_{1\text{goal}}, \theta_{2\text{goal}}) =$ (0.5792, 0.064)·π, † start and goal are swapped).

Further experiments have been carried out and the results are summarised in Table 3.6-2. The tests showed that a valid path could be found after each node has been updated a constant number of times, for updates as in Eq. (3.4-11). The number of sweeps depends on the connectivity scheme as well as on the location of the goal node. Obviously, the number of sweeps does not depend on the size of the grid. Further tests carried out confirm that in any tested environment the number of updates per node using the To&Fro algorithm is below 7 for the grid with 8 neighbours and below 8 for the grid with 4 neighbours. The time to accomplish one sweep is more or less the same for both connectivity schemes, but the 8-neighbourhood connectivity scheme saves one sweep in any start-goal constellation (see Table 3.6-2). It has not been investigated whether a further increase of neighbours (above eight, see [Boult90]) is beneficial in terms of time and, if so, what the optimum number of neighbours is to update the grid in minimum time.

Although the number of sweeps does not change if the grid size is increased, the computational effort increases linearly with an increasing number of nodes. Figure 3.6-4 and Table 3.6-2 clearly indicate that the To&Fro algorithm is of O(N) in time where N is the number of nodes in the grid.

A manipulator experiment which is similar to the above has been carried out (see [Althoefer95e]). The found path has been applied to the physical manipulator, the MA 2000, proving the feasibility of the presented method in a real-world application. The activity distribution of the resistive grid depicted as equi-potential lines is shown Figure 3.6-5. The resulting motion of the manipulator is presented in Figure 3.6-6. This figure has been constructed from snapshots taken by a camera during manipulator motion. The three-dimensional activity distribution of the resistive grid is depicted in Figure 3.6-7.

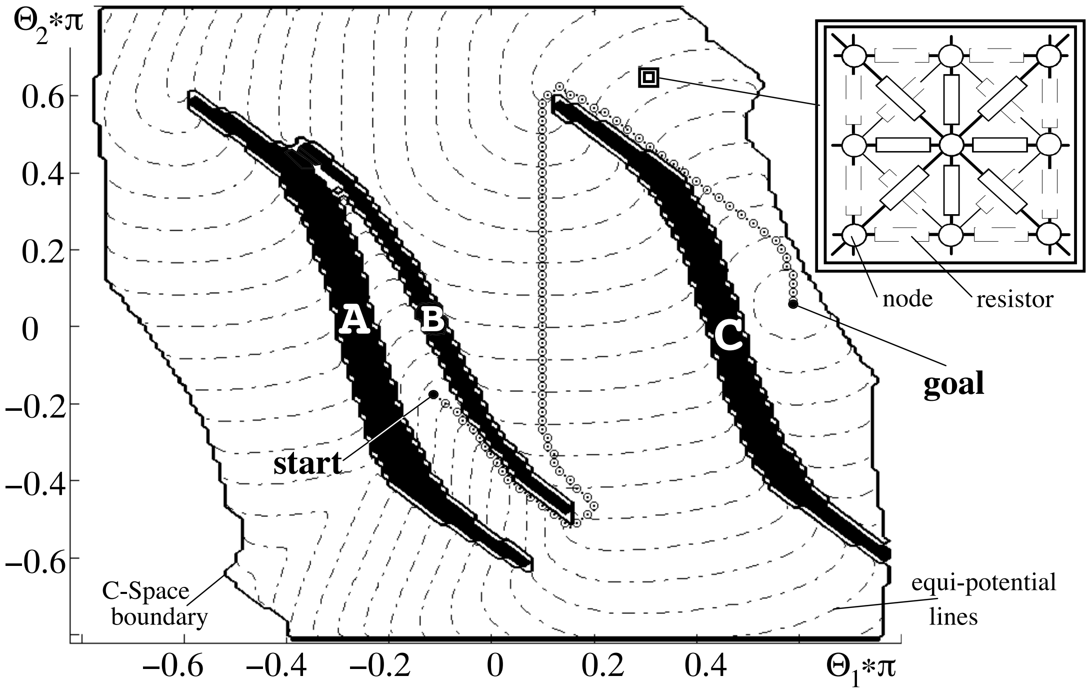


Figure 3.6-5: A gradient ascent performed on the potential distribution generated by the resistive grid guides the robot point from start to goal avoiding C-Space obstacles which correspond to the obstacles “A”, “B” and “C” in Figure 3.6-6. The C-Space boundary is caused by collisions between the manipulator and the base as well as the work bench (see Figure 3.6-6). *Inset*: grid structure (see text for details).

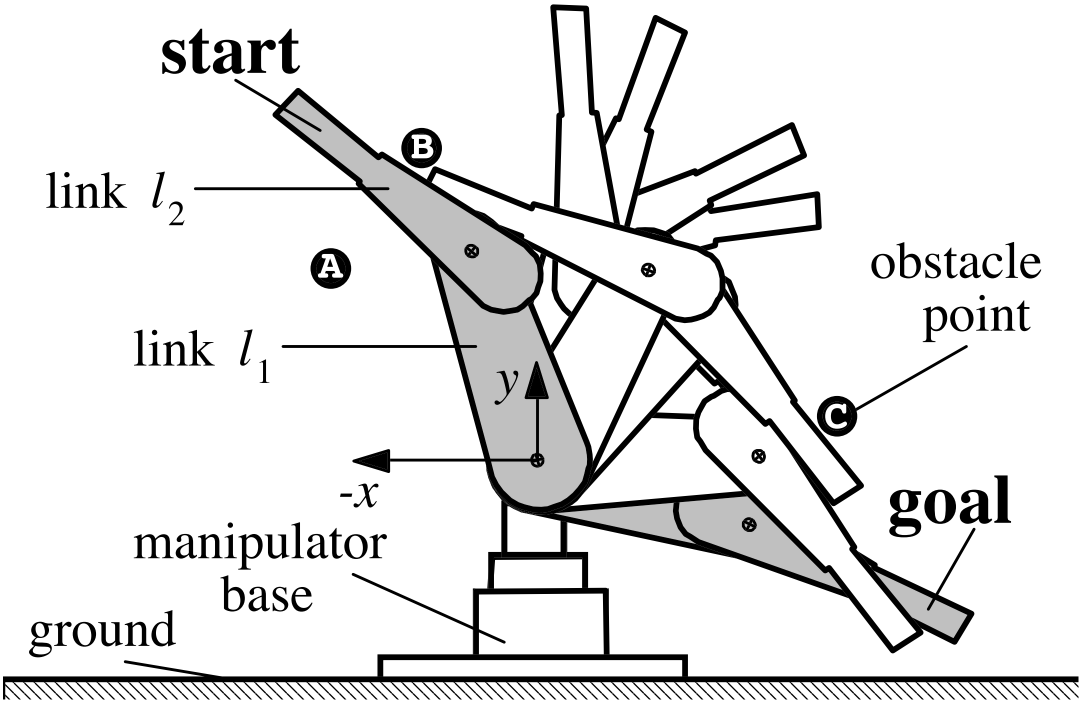


Figure 3.6-6: Depiction of the manipulator (MA 2000) following the path which is generated by the resistive grid shown in Figure 3.6-5.

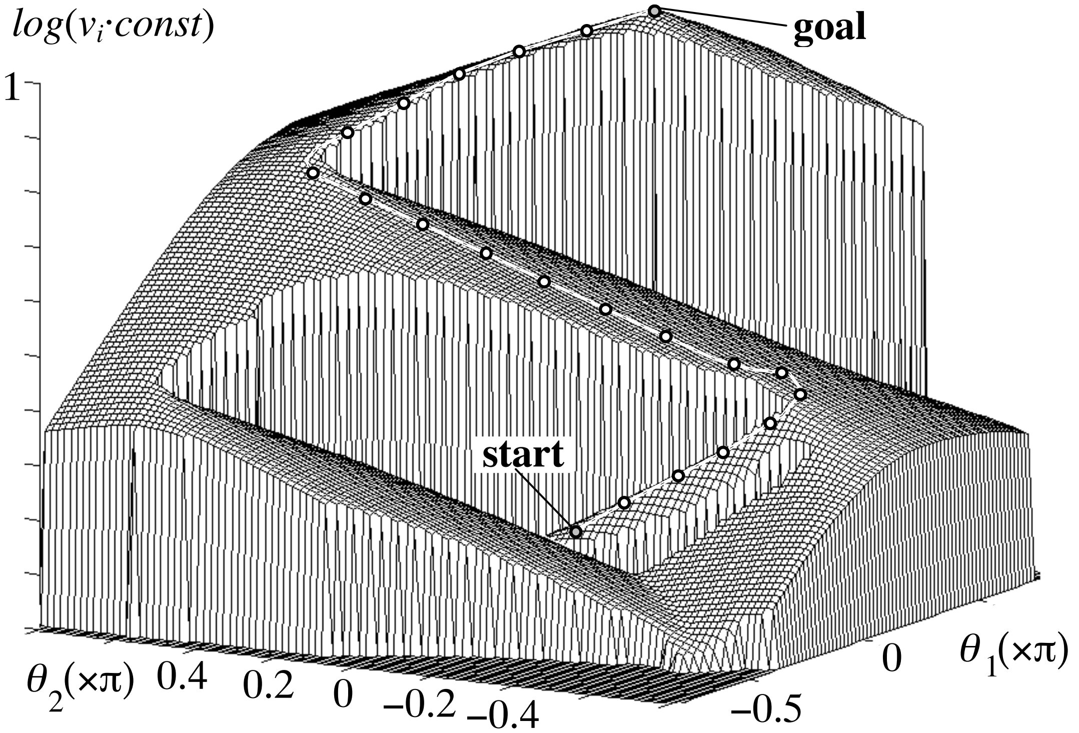


Figure 3.6-7: A 3D-depiction of the activities v_i in the resistive grid after five updates per node.

The activity distribution of the grid due to the To&Fro algorithm which interrupts the update algorithm once the start node perceives a change in activity does not resemble the potential distribution in a physical resistive grid. The activity distribution due to the To&Fro algorithm has been compared to a distribution in a converged grid. Figure 3.6-8 (right) shows the activity distribution of a grid where each node does not experience any further change in activity. Convergence was reached after 159×4 = 636 sweeps, where each sweep goes into a different direction as described in the To&Fro algorithm. The update process was stopped when no grid node changed its activity any further. The maximum change that occurred in the grid before program termination was $2.2959 \cdot 10^{-41}$. To build the converged activity distribution took about 130 times longer than using the early interrupting To&Fro algorithm.

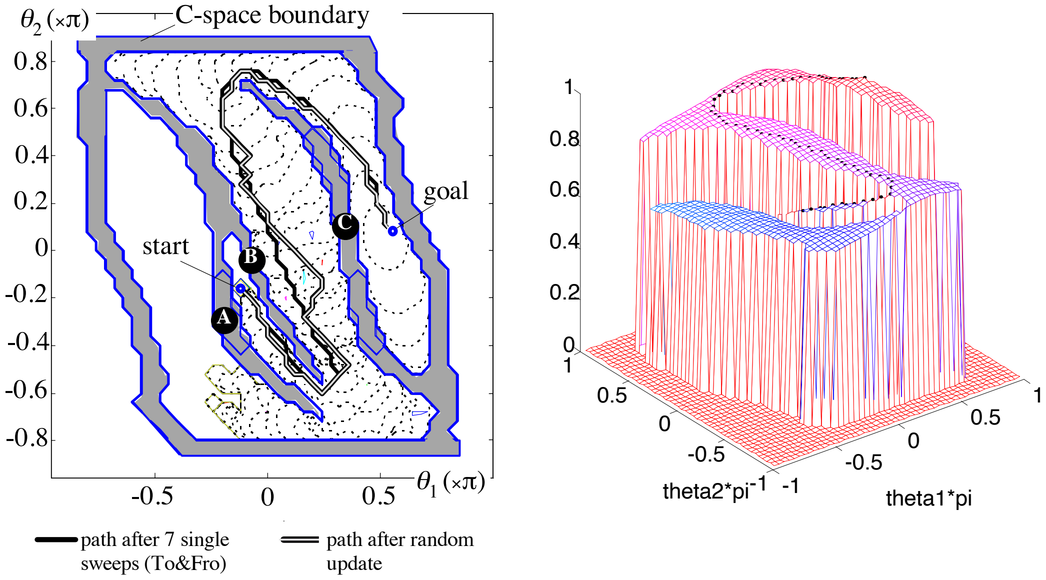


Figure 3.6-8: (left) Comparison of paths found in a converged grid and in a grid whose activity distribution is due to the early interrupting To&Fro algorithm. The grid size is 50×50. The To&Fro algorithm needed five sweeps to produce the shown path. (right) The activity distribution of the converged grid. To build the shown activity took about 130 times longer than using the To&Fro algorithm. Note that the *z*-axis is in a logarithmic scale.

Connolly *et al*. report that the time needed to calculate a zero error solution (converged grid) in a 55×44-nodes grid was 74 seconds. For this solution, 639 iterations were required [Connolly90]. Although the configuration space in their experiment differs from those investigated in this thesis, it can be assumed that the use of the early terminating To&Fro algorithm would provide a big reduction in planning time.

Furthermore, comparing the path found by the To&Fro algorithm with the one found in a converged grid[5], it can be seen that in many cases the differences are minute (see Figure 3.6-8). Also the activity distribution shows only minor differences. From this can be concluded that after only a few sweeps the activity distribution constructed by the To&Fro algorithm represents in many cases a good approximation of the fully converged grid (at least along the constructed path, since other areas are not necessarily updated yet).

[5] In this chapter, grids which are considered to have converged are those grids where the maximum change of activity is below 10^{-5}.

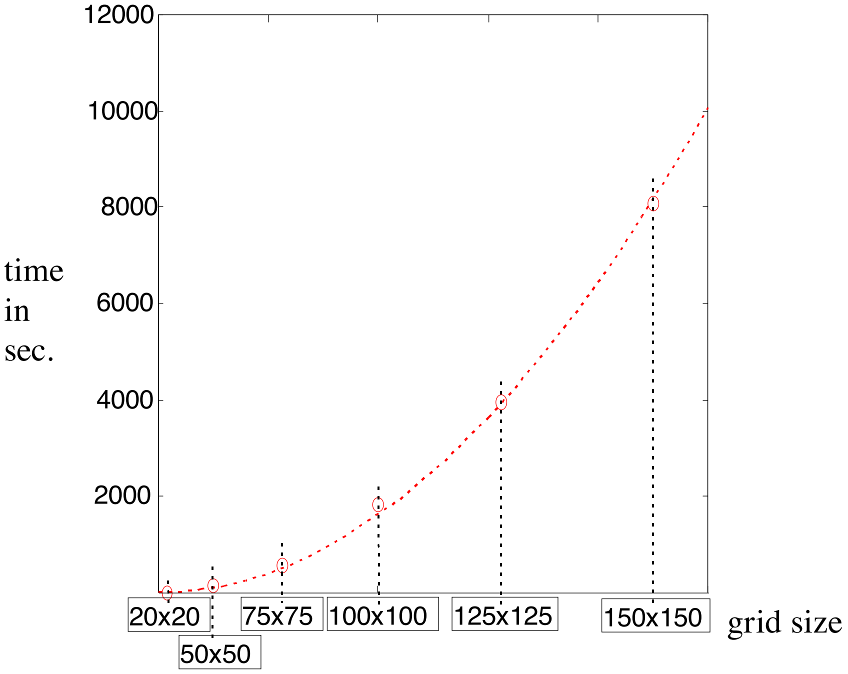


Figure 3.6-9: Convergence times versus different grid sizes. The update process was interrupted when the maximum change of activity was below 1×10^{-5}.

Obviously, the experiments presented here do not suffer from the discontinuity problems discussed in Section 3.5.3. If such problems occurred, the activity distribution would not resemble in all areas an activity distribution found in a converged grid. However, finding a collision-free path is not thus hindered. Further experiments carried out showed that in contrast to the linear increase of time for the To&Fro algorithm, the time to find a converged solution is of $O(N^2)$ (see Figure 3.6-9). This goes along with the estimate given in [Cichocki94]. The To&Fro algorithm provides an activity distribution which allows the construction of a path after a fraction of the updates necessary to build a distribution which approximates the one of a physical grid. This fact makes this novel approach very attractive for robot path planning.

3.6.2 A Planar Three-Link Manipulator

This section describes experiments with “stick-like” planar manipulators as introduced in Chapter 2. This manipulator has three links with revolute joints. Each link has zero width. Link l_1 is 4 units long, link l_2 is 2 units long and link l_3 is 1 unit long. As in the previous sections, path planning in the resistive grid from a given start to a given goal configuration is investigated. Here, the co-ordinates of the 3-dimensional resistive grid describe a 3-dimensional discrete configuration space. For all following experiments in this section, the number of nodes in the neighbourhood of one node is six: two vertical neighbours, two horizontal neighbours and two neighbours above and below. All experiments have been carried out on an IBM-PC-compatible with a Pentium90 using the software package MATLAB. Again, running times are approximately 180 to 200 times slower than in the corresponding compiled C-programs.

In the 3-dimensional grid, eight sweeps are necessary to complete an update cycle according to the To&Fro algorithm described in Section 3.5.2. This means that each node is updated eight times per complete cycle. As it is the case with the two-dimensional problem, the running times are approximately proportional to the number of nodes in the grids (see also Figure 3.6-4).

In the first experiments described here, a path was planned for the three links (l_1, l_2, l_3) of the stick-like manipulator in a planar workspace with two obstacles as shown in Figure 3.6-12. The origin of the workspace is at the centre of the axis of the first link's joint.

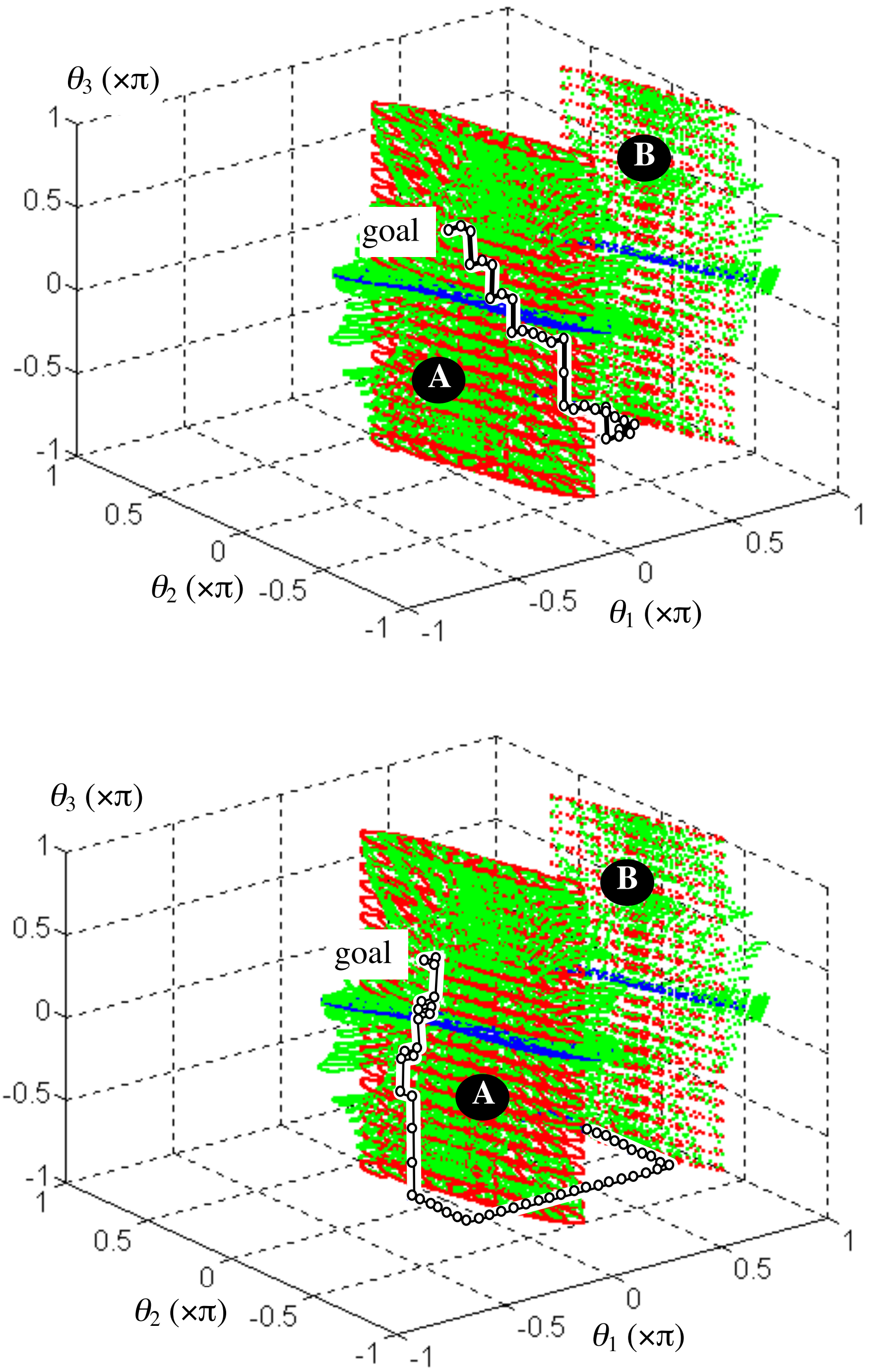
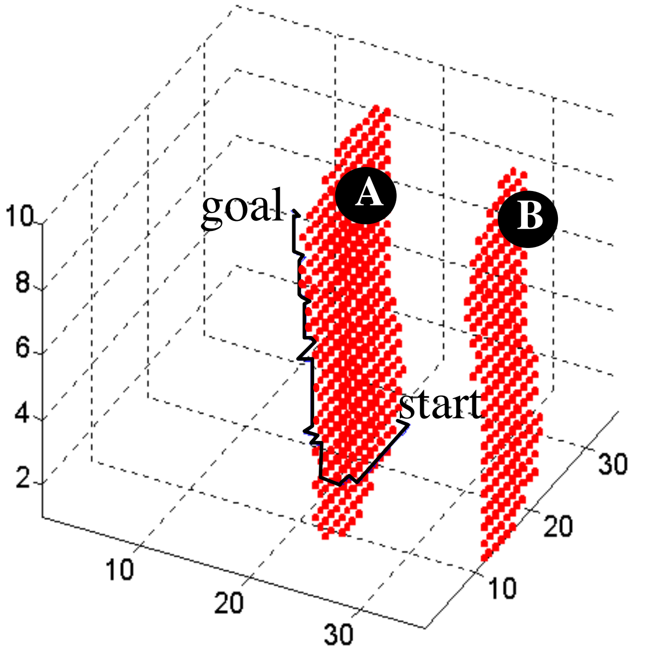


Figure 3.6-10: (top) Path found after 5 sweeps; (bottom) Path found after 1400 sweeps. The obstacles, start and goal configuration were the same in both experiments . The start configuration is hidden behind C-space obstacle “A”.



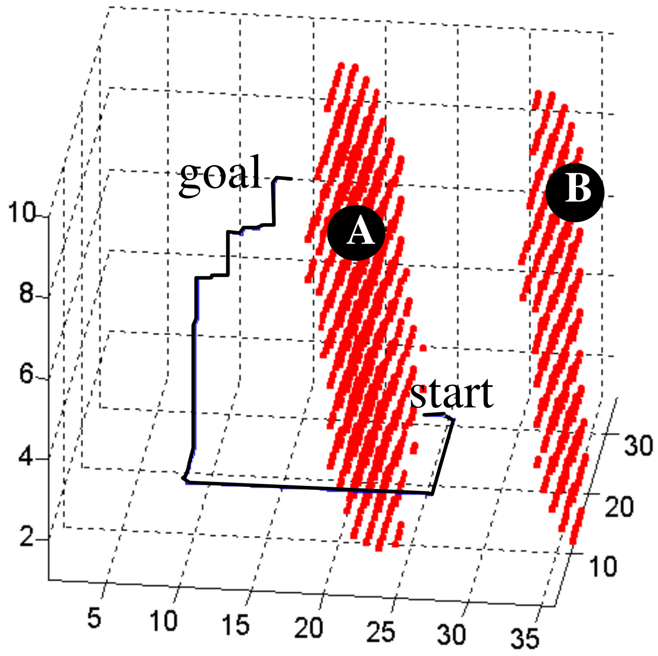


Figure 3.6-11: This figure shows the discrete configuration space as it is applied to the resistive grid. A different view point as in Figure 3.6-10 were chosen to show both start and goal. The top figure shows the path found after 5 sweeps and the bottom figure after 1400 sweeps. Comparing the two paths found by the different methods, it seems as if in the converged grid the path keeps great distance to the obstacle regions. This fact leads to a rather complicated manipulator motion, as shown in Figure 3.6-12 (right).

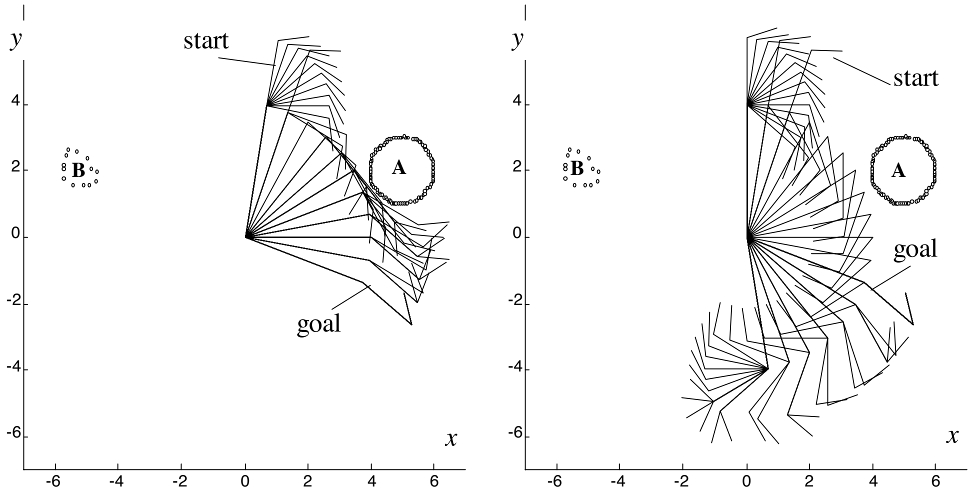
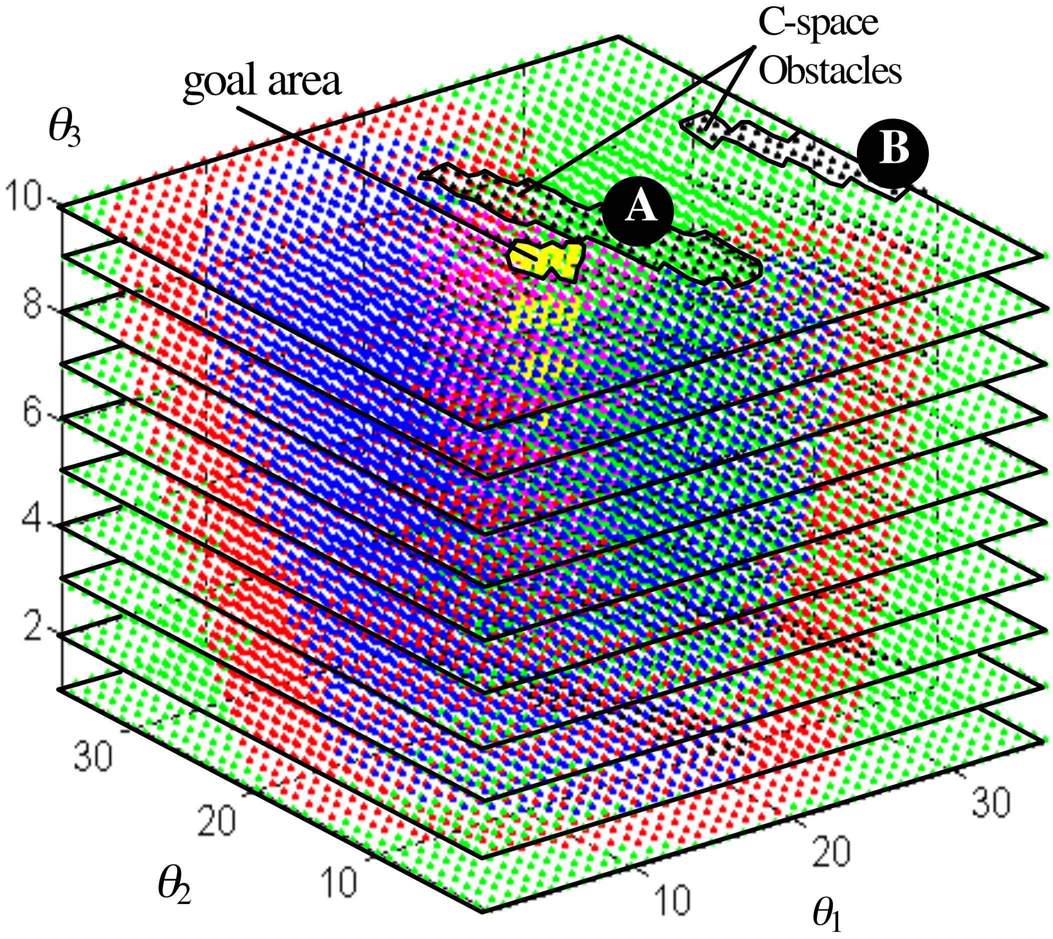


Figure 3.6-12: These figures show the workspace obstacles and the manipulator in motion. (left) Trajectory after 5 sweeps; (right) Trajectory after 1400 sweeps.

Each of the obstacles is described by a number of pixels. The C-space patterns corresponding to these pixels were calculated using the method described in Chapter 2. The calculated C-space patterns are loaded into a map which represents the configuration space in discrete form and each of the map nodes is connected to the corresponding node of the resistive grid (Section 3.4.1). In this experiment, no collision between links was assumed. The working range of each joint was restricted to $[-\pi, +\pi]$ by clamping the nodes along the grid edges to zero activity. The grid size for all experiments was 36×36×10. The following values were chosen as start- and goal-configuration, respectively: $(\theta_{1\mathrm{start}}, \theta_{2\mathrm{start}}, \theta_{3\mathrm{start}}) = (0.4, 0.0, -0.4)\cdot\pi$ and $(\theta_{1\mathrm{goal}}, \theta_{2\mathrm{goal}}, \theta_{3\mathrm{goal}}) = (-.01, -0.1, 0.8)\cdot\pi$ (see Figure 3.6-10 and Figure 3.6-11).

The path shown in Figure 3.6-12 (left) was found after only 5 single sweeps. The execution time under MATLAB was 179.77 seconds. Applying the factor which has been found in view of Table 3.6-1, the path could be found in approximately one second using the corresponding C-program.

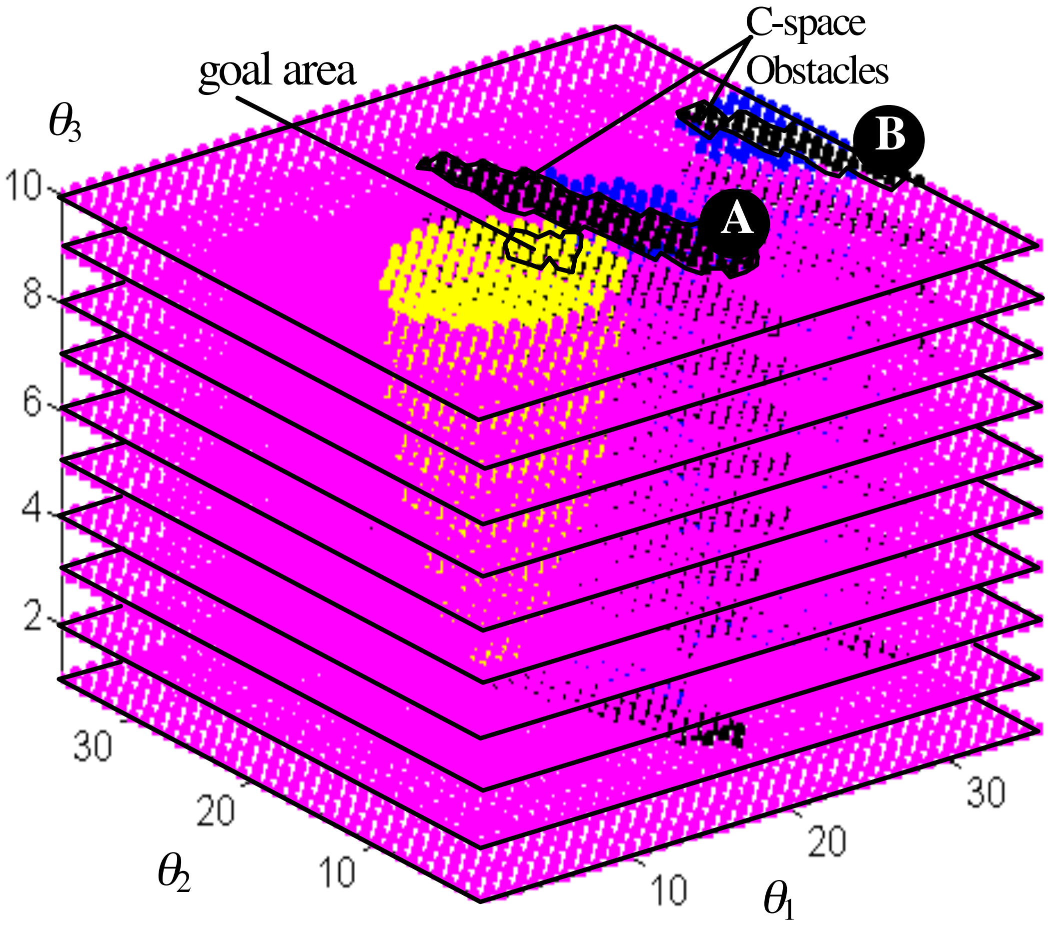
Figure 3.6-13 and Figure 3.6-14 depict the activity distribution of the resistive grid after different update repetitions. Figure 3.6-13 shows the activity distribution using the To&Fro algorithm terminating after only five sweeps. Figure 3.6-14 shows the distribution in a converged grid.



False colour code

black:	$v_i = 0$ (obstacles)	blue:	$10^{-11} < v_i \le 10^{-4}$
green:	$0 < v_i \le 10^{-14}$	magenta:	$10^{-4} < v_i \le .05$
red:	$10^{-14} < v_i \le 10^{-11}$	yellow:	$1 > v_i > .05$

Figure 3.6-13: Depiction of the activity in a 3-dimensional resistive grid after 5 updates per node. The variable v_i represents the activity of node i.



False colour code

black:	$v_i = 0$ (obstacles)	blue:	$10^{-11} < v_i \le 10^{-4}$
green:	$0< v_i \le 10^{-14}$	magenta:	$10^{-4} < v_i \le .05$
red:	$10^{-14} < v_i \le 10^{-11}$	yellow:	$1 > v_i >.05$

Figure 3.6-14: Depiction of the activity in a 3-dimensional resistive grid after 172×8 = 1400 updates per node. The variable v_i represents the activity of node i.

A further experiment was carried out where one obstacle was placed in the manipulator's workspace (see Figure 3.6-16). Since this obstacle is only represented by two pixels the corresponding C-space obstacle is very sparse (see Figure 3.6-15 (left)). To assure that the resistive grid does not construct a path through the C-space obstacle by mistake, the found C-space obstacle has been surrounded by a min-max-box (see Figure 3.6-15 (left)).

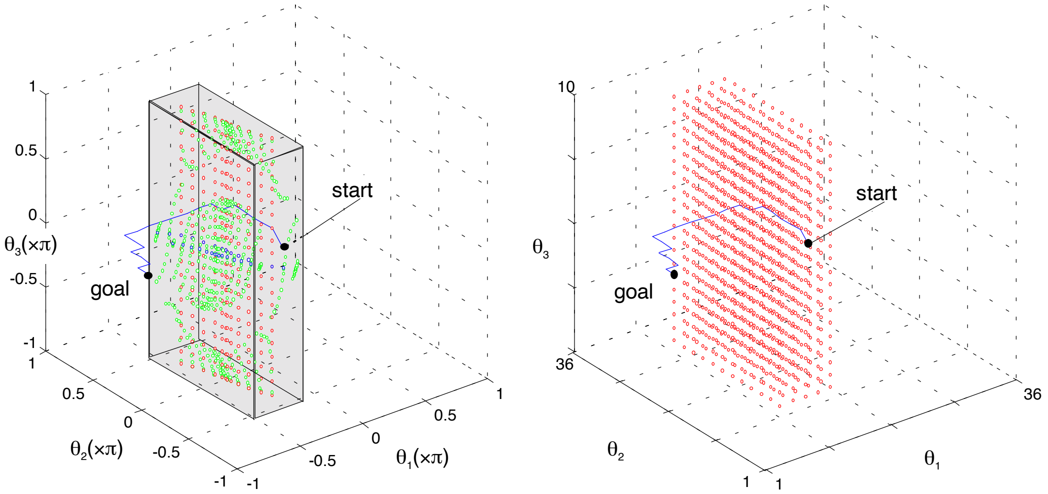


Figure 3.6-15: (left) C-space obstacle with bounding box. (right) Discrete representation of the bounding box.

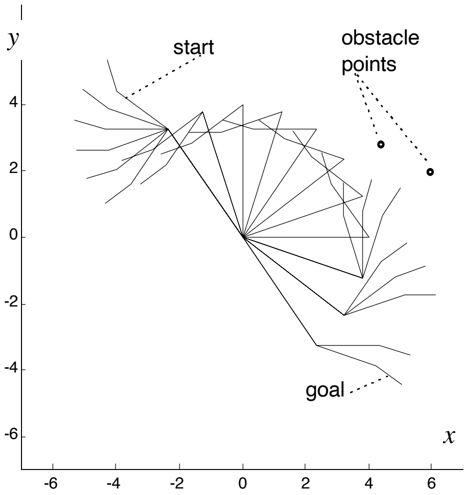


Figure 3.6-16: Depiction of the manipulator moving around obstacle points in workspace.

In this section, the To&Fro algorithm has been employed to build up the activity distribution in a three-dimensional configuration space. Path planning was successfully carried out after only a few update sweeps. Although the method has been only applied to rather simple obstacle constellations, it would find a solution for any possible constellation. In case a complex obstacle constellation is given, this would only require more updates to assure that the activity distribution spreads to a start node which is very distant to the goal node or very much secluded by obstacles.

3.6.3 A Three-dimensional SCARA Manipulator

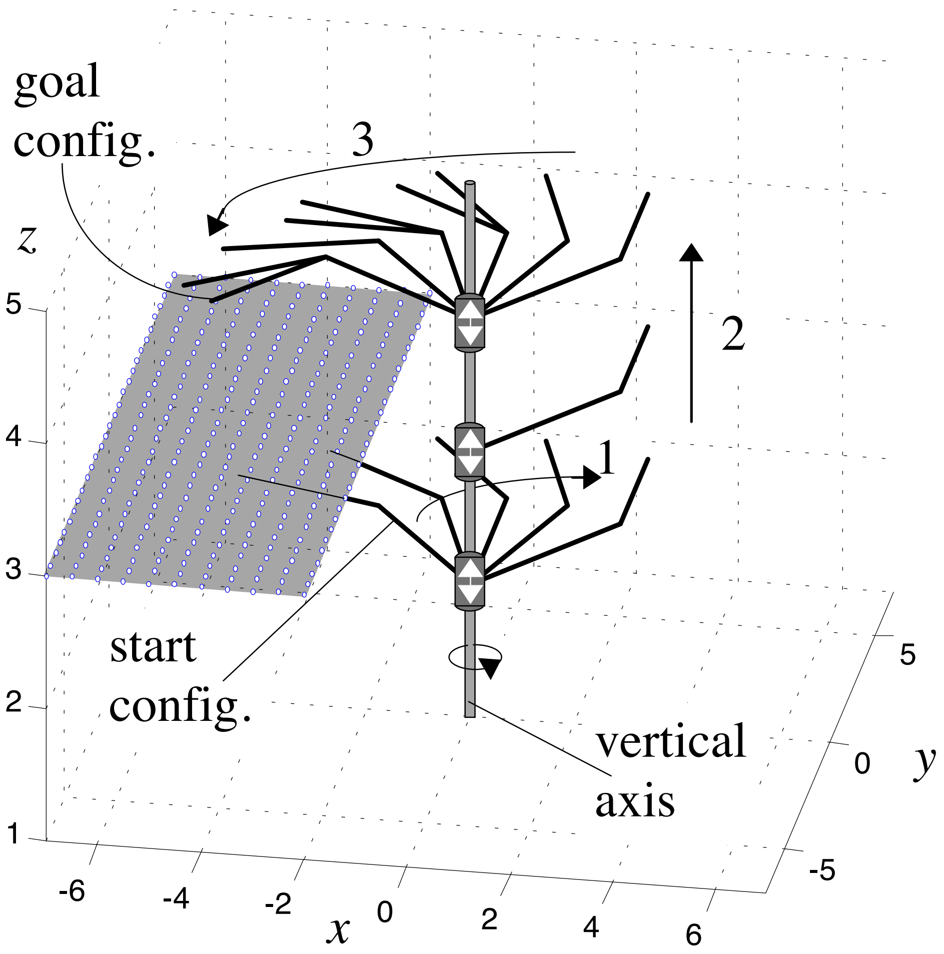


Figure 3.6-17: A three-dimensional SCARA manipulator whose task is to move from the start configuration under the shelf (shaded area) to the goal configuration above the shelf. The labelled arrows next to the links indicate the order of motion. The configuration space has been computed layer by layer employing the transformation technique described in Chapter 2. Each configuration space layer was placed in a 20×20 two-dimensional grid. These grids were combined to form a 5×20×20-grid which was then updated as shown in Section 3.5.2.

To solve a path planning problem for a three-dimensional SCARA manipulator whose two revolute links move in a two-dimensional horizontal plane which is elevated and lowered along a vertical axis (Figure 3.6-17), a three-dimensional resistive grid has been used to compute the path. The three-dimensional configuration space has been constructed by computing two-dimensional C-spaces for each of the five horizontal layers in *z*-direction (Figure 3.6-17) employing the technique described in Chapter 2 (see also [Newman91]).

3.6.4 A Mobile Robot in a 3D-Workspace

This section shows the use of the resistive-grid planner to solve a 3D mobile-robot path-planning problem.

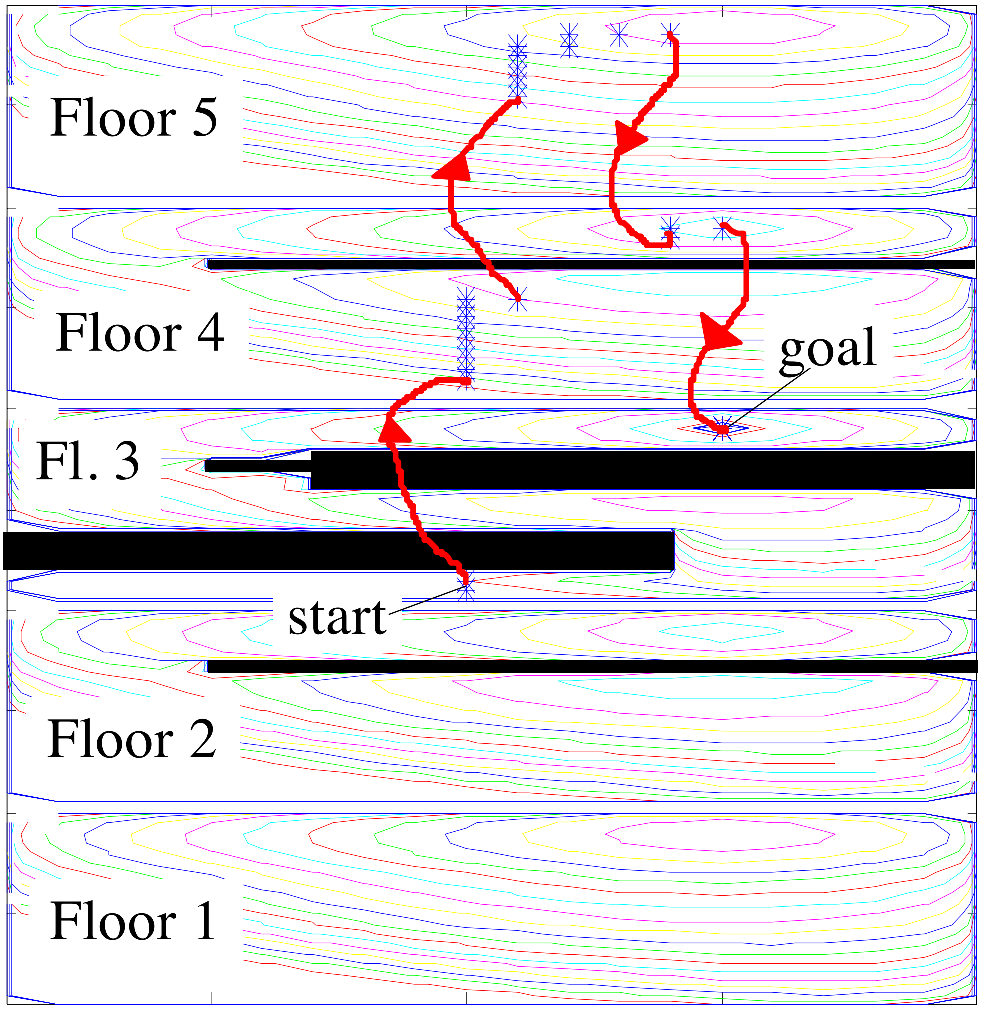


Figure 3.6-18: This figure shows the 3-dimensional workspace of a point robot. Obstacles are depicted by black rectangles. The path is planned using a 3-dimensional resistive grid. The point robot can move between 5 different floors. The stars show the path found by the resistive grid from start to goal. Whenever the robot changes from one floor to another, arrows are used to depict this change. Note that even though start and goal are on the same floor (Floor 3) and a possible path exists on this floor, a shorter path is found by the resistive grid going via other floors. The curvy lines are the equi-potential lines which develop around the goal node. The boundary condition is the Dirichlet one. The To&Fro algorithm was used for updating the grid.

3.7 Comparative Studies

3.7.1 Comparisons to Other Update Rules

Apart form the resistive grid approach which makes use of the update rule described in Eq. (3.4-11), other update equations which compute an activity distribution over a state space are conceivable. Two update equations which can be found in the literature are presented here [Cooper81, Siemiatkowska94b] and compared to the resistive grid approach. Both update equations are based on the theory of dynamic programming [Bellman57, Cooper81]. The first uses an additive cost function, while the second uses a product cost function. The latter has been also employed by Siemiatkowska *et al*. for mobile robot path planning. They describe their approach as a diffusion process [Siemiatkowska94b]. Instead of using an iterative method, as suggested in this chapter, to spread the activity distribution, Siemiatkowska *et al.* make use of an approach which is similar to the building of a cost-function tree as described for the A*-algorithm in Section 3.7.3.

In the approaches based on dynamic programming, each transition from one node to another is assigned with a local cost. The goal is set to a high value, here: 1. Obstacles are set to a low value, here: 0. In case of the additive function, any free node's activity changes during the update process to a new activity which is the result of the subtraction of a constant from the maximum of the neighbours' activities (In this experiment the additive constant was chosen to be -0.001). In case of the product function, any free node's activity is substituted by the product of a constant and the maximum of the neighbours' activities (In this experiment the factor was chosen to be 0.8). Note, that in the approach which uses the additive cost function the constant has to be chosen carefully to assure that under no circumstances a node is assigned an activity which represents obstacles. Otherwise, originally obstacle-free nodes are interpreted as obstacle nodes.

In all four cases the To&Fro algorithm has been employed to spread the activity. The update process is interrupted once the activity has spread to the start node.

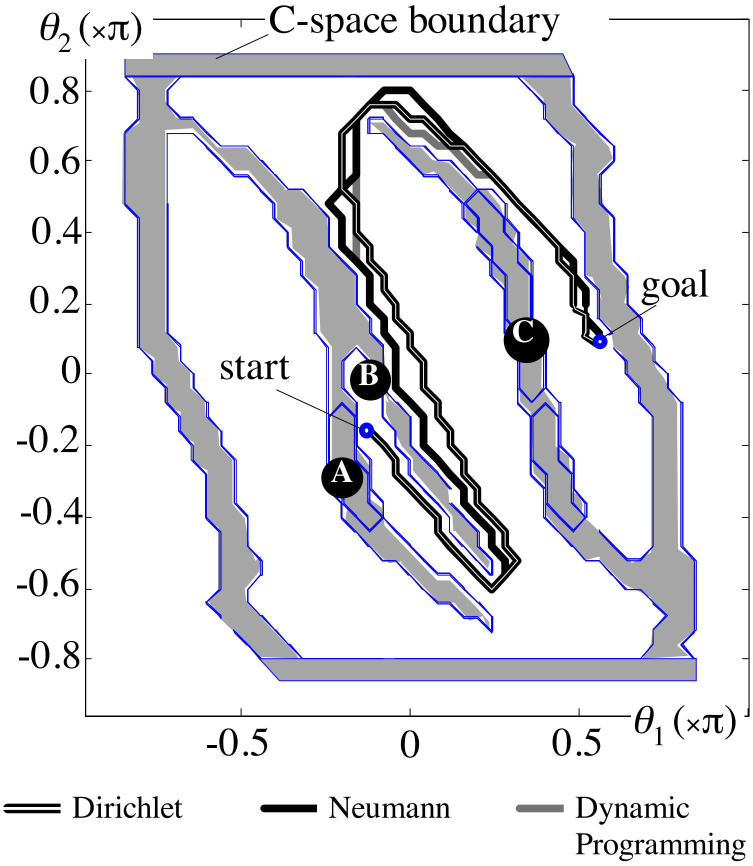


Figure 3.7-1: Comparison of paths found by four different approaches: a) resistive grid using Dirichlet's boundary condition, b) resistive grid using Neumann's boundary condition, c) dynamic programming using an additive cost function, d) dynamic programming using a product cost function. The paths found by c) and d) are identical. In all cases the To&Fro algorithm was used to spread the activity. The update process was interrupted when the start node or its neighbours experienced a rise in activity.

In this particular experiment, the number of neighbours per node were four. Path finding was not restricted along horizontal and vertical neighbours only, but was additionally allowed to proceed along diagonal neighbours. However, in all four cases the path had a length of 67 units and proceeds along a very similar course. It shows that the updating times are very similar. The resistive grid with the Dirichlet boundary condition performs slightly faster than the other approaches, while the grid using the Neumann boundary condition is the slowest. Note that in all four cases the especially adapted To&Fro algorithm has been used to spread the activity distribution. If a different update sequence is used, the execution times can be much higher as shown in Section 3.7.2.

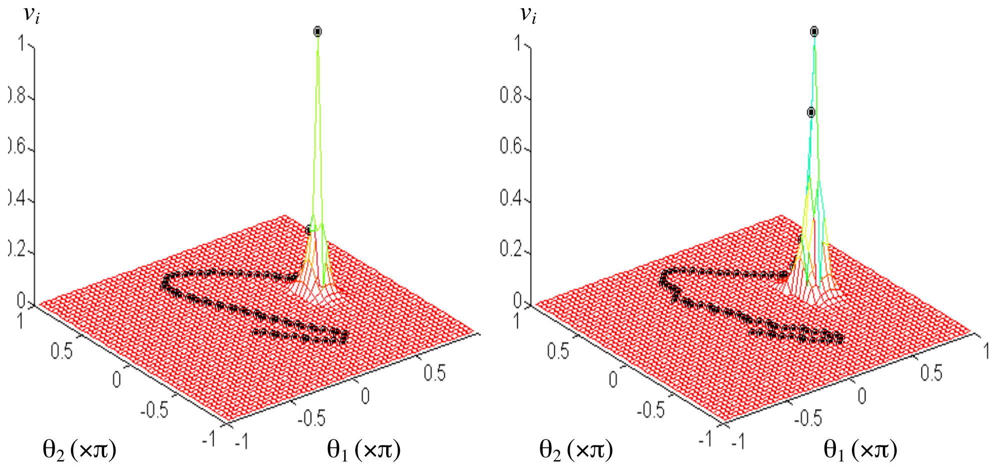


Figure 3.7-2: (left) Activity distribution found in the resistive grid using the Dirichlet boundary condition (execution time: 9.06 sec.); (right) Activity distribution found in the resistive grid using the Neumann boundary condition (execution time 17.41 sec.).

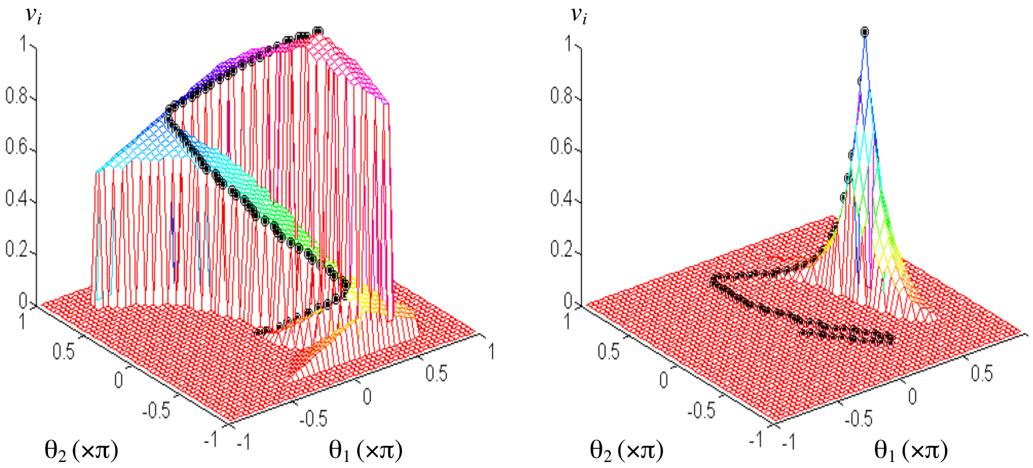


Figure 3.7-3: (left) Activity distribution found by the dynamic programming algorithm with the additive cost function (execution time 14.15 sec.); (right) Activity distribution found by the diffusion algorithm (product cost function) (time 12.8 sec.).

3.7.2 Comparison to Other Update Sequences

Other sequences for updating the rule in Eq. (3.4-11) have been investigated. These update sequences do not make use of the specific shape of the C-space patterns occurring during manipulator path planning. The investigated sequences are the unidirectional and the random update. These update methods were found to be much slower than the proposed one (see Table 3.7-1). Whichever of the two update sequences one uses, they take about 7 to 8 times longer than the To&Fro algorithm to construct an activity distribution along which a path can be found. All update sequences shown in Table 3.7-1 are “sweeping” the entire grid independent of the locations of start, goal and obstacles. They are of order O(N) in time.

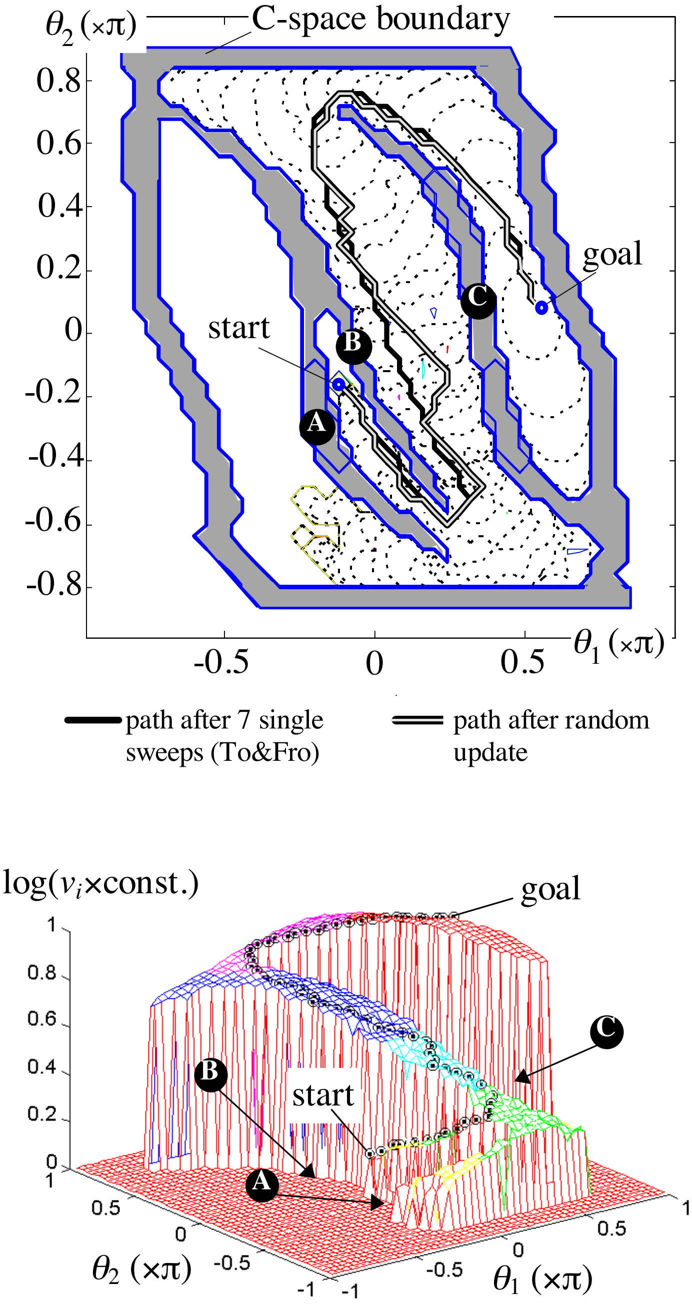


Figure 3.7-4: Activity distribution in a resistive grid which had been randomly updated. The grid size is 50×50. The shown distribution was reached after 80.74 sec.

In contrast to the random update, the To&Fro algorithm produces quickly a distribution which is well-spread over the whole resistive grid (see Figure 3.6-7 for comparison), while the path found by the random update procedure was very jagged (see Figure 3.7-4).

random update	unidirectional update		To&Fro
average update time in sec.	sweep direction	time in sec.	time in sec.
71.46 (averaged over 12 experiments)	1	78.38	9
	2	74.54	
	3	66.02	
	4	61.76	

Table 3.7-1: Comparison of different update orders. The chosen grid size was 50×50. The update process was in any case terminated once the start node experienced a rise in activity. The execution times were determined from routines running under MATLAB.

3.7.3 Comparison to the A*-Algorithm

The shortest path is the one with the minimum overall cost. In a discrete grid where nodes are placed equidistantly to each other and every node-to-node connection is associated with the same weight parameter, the shortest path between a start point and a goal point is the path with the shortest Manhattan (or city-block) distance. The shortest path in the Euclidean sense cannot be calculated, employing an optimisation process which "knows" only about discrete steps in horizontal and vertical direction. Obviously, due to the nature of discrete distances, there may be many different routes that have the same overall cost and the same length [Boult90].

The A*-algorithm is a dynamic programming method (see Section 3.2.4). Although, it is usually used for graph searching (in the visibility graph, cell decomposed maps, etc.), it can be also applied to the presented problem, since path planning in a discrete configuration space can be also viewed as graph search problem. Each node in the grid can be viewed as node in a graph. Each node is connected via arcs to neighbouring nodes. Each arc is assigned with a certain cost which is associated with the "difficulty" to cross the arc and which is here proportional to the Euclidean distance between two connected nodes.

Beginning with the start node, the A*-algorithm calculates the minimal cost between adjacent nodes and start node. This process is continued by visiting the adjacent nodes and calculating for each of those the minimal costs between their neighbours and the start node. This process is further continued and a tree is built in which each visited node is assigned with a pointer to its neighbour which has the minimal cost to the start node. During this process, nodes might be found that point to the "wrong" neighbour while another neighbour has a smaller cost due to a "newly" found path segment which provides now a less costly connection to the start node. (This is similar to the discontinuity problem discussed in Section 3.5.3.) Then the wrong pointer is redirected to the neighbour with the smallest cost.

In contrast to other dynamic programming methods (see Section 3.2.4), the A*-algorithm does not assign a pointer to every node, but only to those nodes it visits until a solution which connects start with goal is found. The A*-algorithm terminates when finding the goal node or after all nodes have been visited and the goal node was not among those. The found path is the least costly path between start and goal node. In the path planning problem discussed here, the found path represents the shortest path. As already stated, there may be two or more different routes that have the same overall cost in this discretised state space [Boult90].

start	goal	A*	start	goal	A*
co-ordinates		(time in seconds)	co-ordinates		(time in seconds)
18, 18	2,2	4.45	2,2	18, 18	4.5
18, 18	4,4	4.23	4,4	18, 18	4.34
18, 18	6,6	3.63	6,6	18, 18	4.23
18, 18	8,8	2.75	8,8	18, 18	4.23
18, 18	10,10	1.76	10,10	18, 18	4.23
18, 18	12,12	0.83	12,12	18, 18	3.03
18, 18	14,14	0.33	14,14	18, 18	0.98
18, 18	16,16	0.11	16,16	18, 18	0.22
18, 18	18,18	0	18,18	18, 18	0
		To&Fro			To&Fro
		(time in seconds)			(time in seconds)
any distance		0.27	any distance		0.83

Table 3.7-2: Comparison of running times under MATLAB for the A*-algorithm and the To&Fro algorithm in an obstacle-free grid. The different start and goal nodes are described by their grid co-ordinates.

The A*-algorithm has been compared to the To&Fro algorithm. In the experiment here, both algorithms have been applied to find a path between a start node and a goal node in an obstacle-free state space only confined by edges which are clamped to zero. The grid size was 20 by 20 nodes. Each node is connected to four neighbours. Path searching is restricted along horizontal or vertical directions.

Different start and goal nodes have been assigned (see Table 3.7-2). It can be seen that the A*-algorithm's search time depends on the distance between start and goal, while the To&Fro algorithm is independent of this distance. Especially for long distances the To&Fro algorithm is superior to the A*-algorithm.

The A*-algorithm has been also applied to the manipulator planning problem which had been employed already in earlier experiments. The results for a 50×50-grid with the 4-neighbourhood scheme are shown in Figure 3.7-5. Again, path finding is only allowed along vertical and horizontal neighbours. Two experiments have been carried out. The first is depicted in Figure 3.7-5. For the second experiment goal and start position were swapped. In both cases, the To&Fro algorithm found a path in shorter time. (First experiment: To&Fro algorithm - 14.61 sec., A*-algorithm - 19.96 sec.; second experiment: To&Fro algorithm - 11.03 sec., A*-algorithm - 29.8 sec.) The difference in time is presumably related to the way the activity distribution or the cost-look-up table is built. The effectiveness and speed of the To&Fro algorithm is due to its simplicity. A fast average mechanism sweeps over the entire grid regardless of goal location or obstacle constellation. The A*-algorithm, on the other hand, makes use of complicated routines which spread over the obstacle-free areas of the grid by "wandering" along the outside of obstacles and placing nodes according to their distance to the goal into a tree-like list. The building of this list seems to be time consuming.

The drawback of the resistive grid using the early interrupting To&Fro algorithm is that it does not always find the shortest path (see Section 3.5.3). However, in the experiments here the length of the path obtained by all three approaches (A*-algorithm, To&Fro algorithm, converged grid) was the same (102 units). This means that in these experiments the resistive-grid-based approach found a shortest path. The difference between the obstacle constellation shown here and discussed in Section 3.5.3 is that in the experiments here the activity distribution cannot spread around any obstacle from two sides and thereby form a discontinuity. Even if the path found by the resistive grid is a sub-optimal in terms of length, it is always a collision-free path which guides the arm successfully to the goal location.

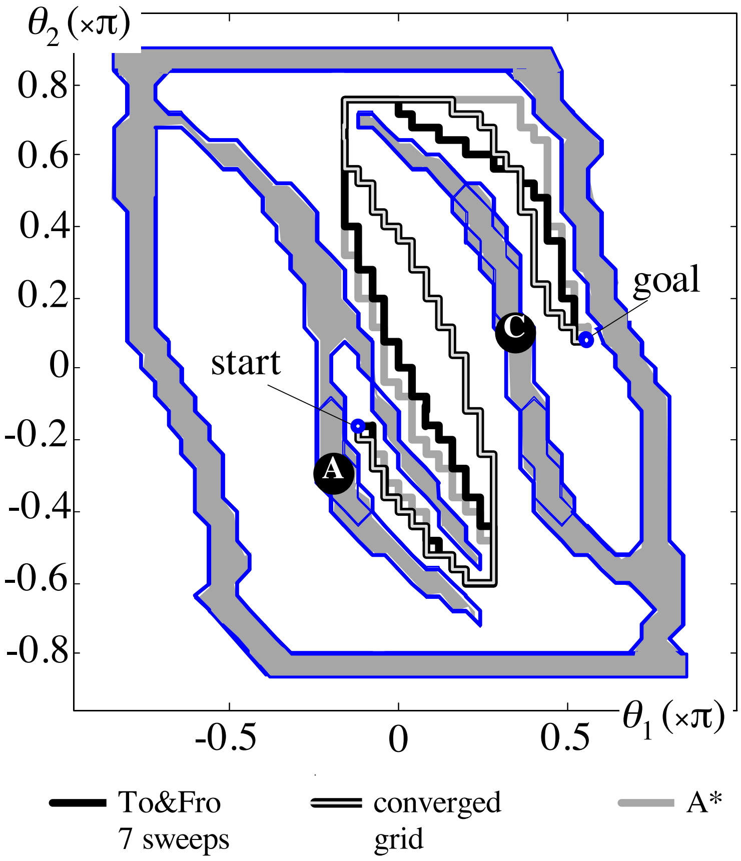


Figure 3.7-5: Comparison of three planning strategies: To&Fro algorithm, converged grid and A*-algorithm. In this grid of size 50×50, each node is connected to four neighbours and path search is restricted to vertical and horizontal directions. The start- and goal-configuration are $(\theta_{1\text{start}}, \theta_{2\text{start}}) = (0.5792, 0.064)\cdot\pi$ and $(\theta_{1\text{goal}}, \theta_{2\text{goal}}) = (-0.111, -0.1796)\cdot\pi$. Further details see text.

3.8 Summary

In this chapter the application of a computer-emulated neuro-resistive grid to solve the global manipulator path planning problem has been investigated.

The neuro-resistive grid incorporates ideas of the conventional resistive grid as well as Hopfield neural network type. Resistive grids have been only recently introduced as means for path planning [Connolly90, Tarassenko91]. So far resistive grids for path planning have been investigated on a theoretical level and on the base of simulations [Bugmann95, Connolly90, Tarassenko91]. Neural-network-based approaches have been applied to simulated robotic problems [Glasius94] and to mobile robots [Siemiatkowska94b, Kanaya94]. Among other things, this chapter expands on this and shows the feasibility of the suggested method in real-world manipulator applications.

A new update algorithm, the To&Fro algorithm, has been proposed which rapidly generates an activity distribution over the obstacle-free areas of a manipulator's configuration space. This activity distribution can be exploited to construct a path from a start node to a goal node by following the greatest gradient. The update process is terminated when the activity distribution has reached the start node. It has been shown that the new algorithm spreads the activity distribution in the grid approximately eight times faster than other update sequences found in the literature [Cichocki94, Glasius94]. Moreover, the execution time for the To&Fro algorithm increases only linearly with the number of grid nodes while the time necessary to reach a converged grid is of $O(N^2)$, where N is the number of grid nodes [Cichocki94, Connolly90]. This aspect makes the To&Fro algorithm very suitable for path planning with strong real-time constraints.

A converged resistive grid approximates the calculation of harmonic functions on a confined, connected region satisfying Laplace's equation [Connolly90, Ramo94, Noble64]. The solution found in a converged grid is unique (see [Ramo94]) and depends only on the shape of the boundary [Glasius94]. This chapter has shown that the To&Fro algorithm, although it does not satisfy Laplace's equation due to its early termination of the update process, produces also a local-extrema-free activity distribution which is well suited to solve the path planning problem in robotics always providing a collision-free path from a start state to a goal state - if such path exists.

A comparison between the two boundary conditions showed that a computer-emulated grid with Dirichlet's boundary condition produces a suitable activity distribution faster than a grid with Neumann's boundary condition.

The To&Fro algorithm has been also compared to the A*-algorithm, which is a global graph search method often employed in robotic planning tasks to find a shortest path [Latombe91, Cires96]. The To&Fro algorithm, whose execution time is to a great extent independent of the location of obstacles regions, goal node and start node, is faster than the A*-algorithm in C-space constellations where the goal is very distant to the start. Although a path found by the To&Fro algorithm is in some situations not a shortest one, it is always collision-free. Considering that the focus of this chapter was on finding collision-free paths in short time, the To&Fro algorithm outperforms the A*-algorithm.

A further extension to the basic neuro-resistive grid has been developed which allows path planning in non-homogenous and non-topologically ordered graphs and can also be used to apply soft safety margins to obstacle regions. In metric maps the gradient between two nodes can be usually derived by computing the difference between the activities of two nodes. In grids where the weight factors or resistors between nodes are not constant, the gradient is the division of activity difference and the connecting weight factor. Incorporating this aspect, the resistive grid has been successfully applied to a general graph search problem and to a robot planning problem where obstacle regions have been expanded by a soft safety margin.

Chapter 4

Fuzzy-Based Navigation and Obstacle Avoidance for Robotic Manipulators

This chapter describes a novel navigation and obstacle avoidance method for robotic manipulators, called here fuzzy navigator, which is based on fuzzy logic. In contrast to the global planning strategy presented in the previous chapter, obstacle avoidance as well as local navigation methods are methods in which calculations are not based on the entire state space but only on a confined region surrounding the actual state. Local methods like the one suggested here outperform global methods in many situations and applications. Owing to their simplicity and hence their short response time, local methods are especially suitable in on-line applications with strict real-time requirements. Furthermore, these methods allow obstacle avoidance in uncertain environments and can be used in safety critical applications. In contrast to many global planning strategies, they can control autonomous robots and robotic manipulators with a high degree of freedom (DOF). The main disadvantage of these local methods when compared to a global planning strategy is that they can get stuck in dead-lock situations, thus, their inability in some cases to recognise a possible solution to a problem.

This chapter is organised as follows: Aspects of local navigation and a general description of the proposed navigation method are presented in Section 4.1. The potential field method and other obstacle avoidance methods which had been influential to this work are outlined in Section 4.2. Section 4.3 gives a detailed description of the fuzzy-based navigation and obstacle avoidance method for robotic manipulators. The technique has been applied to simulated manipulators as well as to a real one, as shown in Sections 4.4 and 4.5, respectively. Initial experiments regarding the training of the fuzzy navigator are described in Section 4.6. Section 4.7 concludes this chapter.

4.1 Problem Definition and System Overview

Local navigation and obstacle avoidance become a necessary engineering task when the path cannot be computed prior to the robot motion either because of unforeseeable changes in the environment or missing information about obstacle locations.

Traditionally, obstacle avoidance and local navigation techniques are part of multi-layered planning systems. Each layer represents a functional module carrying out a specific task. The modules have usually functions, such as task planning, global path planning, obstacle avoidance, motor control. A possible approach is a top-to-bottom hierarchical structure where modules on a higher level control those on a lower level [Althoefer94b]. Those planning systems provide the robot with great independence. The users of such systems do not need to know about the detailed commands sent to the robot's actuators. Instead, they only need to provide a general and abstract task the robot is supposed to carry out. In a self-reliant manner, the system turns the user command into the appropriate robot motion. This process involves interpreting the task and transforming it into locations which have to be reached during execution of the task. This, in turn, demands a global path planning strategy to be invoked and to plan routes between those locations, thereby avoiding obstacles in the given workspace. Obstacle avoidance and local navigation techniques are then used to adapt the planned path to changes in the environment.

The output of the global planner can be interpreted as a sequence of path segments or via-points. Usually simple trajectories, such as straight lines, connect these via-points. Changes in the environment may cause a straight connection from start to end of the path segment to be inaccessible at certain times. Thus, one via-point after another is given to the local navigator and its task is to traverse towards the next via-point thereby avoiding any obstacle. Since the navigator cannot foresee all possible situations which will be encountered by the robot, the navigator must be able to abstract and generalise [Tschichold96, Sulzberger93]. Highest priority is usually given to obstacle avoidance, thus, even though the goal or via-point cannot be reached, the robot stays away from obstacles.

While path planning and obstacle avoidance in regard to mobile robots is commonly only concerned about the control of a single object, manipulators consist of a chain of objects and for each such object or link it must be guaranteed that no collision with obstacles occurs at any time. Thus, the obstacle avoidance technique must provide a collision-free route for all links. Since the motion of some links does not occur independently of the motion of other links, these interactions have to be taken into account by the obstacle avoidance technique (see also Section 4.3.2). The envisaged fuzzy-based technique can be also utilised in cases where the path planner or an operator (via teach pendant) provides a desired end effector motion [Risse95]. In such an implementation example, the technique shows a purely obstacle avoidance behaviour clearly supporting the global planning strategy provided by the operator.

In all situations where obstacle avoidance or navigation problems have to be solved, sensors play a key role. Multiple sensors are commonly used to provide the planning system with sufficient information about the environment. Examples include cameras, encoders on steering and drive mechanisms, inertial navigation systems, close contact sensors, acoustic and optical sensors, infrared and laser rangefinders [Brooks86, Latombe91, Drews92, Lee96, Freund96]. Since the focus in this chapter is on the algorithm which provides obstacle avoidance and local navigation, the properties of sensors (such as errors in sensor readings, sensor data fusion, etc.) are not further investigated here. In this chapter, the models used for sensors (for example, ultra-sonic sensors mounted on the manipulator links [Risse95]) are assumed to work without error and therefore, provide at all times an exact picture of the environment. Nevertheless, the great robustness which the fuzzy control system has shown during experimental studies suggests that data which is produced by real sensors could be also dealt with in a satisfactory way.

The navigation technique presented in this chapter is based on fuzzy logic. A fuzzy logic system or fuzzy system is a universal approximator which provides a rule-based mapping between the input and the output space, while classical approaches make use of analytical functions to solve the navigation problem (see for example Section 4.2.1). In this particular application, the input space is defined by axes which represent the distances to nearby obstacles (for example acquired by ultra-sonic sensors) as well as the error between actual and desired goal configuration. The output or command space is defined by axes which represent the command variables. These are commands which drive the actuators of the manipulator links. The suggested system is divided into separate fuzzy-units which control each manipulator link individually. Thus, each of these fuzzy units reacts with a single output variable, the new actuation command for the corresponding link. The actuation command is computed in response to the unit's inputs which induce *i*) a repelling influence based on the link's distance to nearby obstacles and *ii*) an attracting influence based on the difference between the link's actual state and its target state.

The mapping provided by the fuzzy system is usually based on the knowledge of a human expert. The expert knowledge can be used by the system designer to construct the fuzzification stage, and the defuzzification stage as well as the rule base (for further details see Section 4.3.3). In other words, the designer sculpts a function by adjusting a set of parameters in order to approximate a desired mapping. These parameters can be further refined by trial and error (see Section 4.4) or by neural network training techniques (see Section 4.6 and [Kosko92, Tschichold96b, Keerthi95, Berenji92, Jang92]).

The response of the fuzzy navigator is a reactive one and generates an actuator command at each iteration. Each command generation is carried out irrespectively of past or future events; only the currently available data inputs are considered. Thus, path planning which produces a sequence of path segments is not carried out by the fuzzy navigator. In the case of robot navigation in a cluttered environment, the ideal mapping leads the robot to the target without causing collisions with obstacles. The actual mapping implemented in the fuzzy system will always be an approximation to this ideal one. This is due to the fact that the mapping is confronted with a variety of different tasks. The main design considerations in regard to the navigation problem can be described as follows: The mapping has to react properly to a local constraint, like for example increasing the distance to an obstacle if the distance falls short of a given threshold. However, it is not sufficient to optimise the mapping for one particular instant, since during motion the robot is faced with many different situations which demand a variety of different reactions. Thus, the designer's task is to tune the system in such a way that the fuzzy mapping integrates the different aspects of the control problem at hand in the best possible way.

The fuzzy navigator presented here is especially adapted to the underlying problem of steering a robotic manipulator as opposed to steering a mobile robot which can be viewed as a single moving object. The motion of the links of a non-branching manipulator cannot be dealt with in a completely separate fashion because the motion of some links influence the motion of others. To meet these requirements, those fuzzy units, which are connected to *distal* links[1], communicate information about their relation to their nearby environment to units of *proximal* links[1] (for further details, see Section 4.3.2).

There are many aspects that make the fuzzy system proposed here superior to other navigation techniques or path planning strategies. Its short response time allows the fuzzy navigator to provide quickly a new actuator command and makes it therefore suitable for on-line implementation. Dynamic and uncertain environments can be successfully dealt with. In contrast to global path planning methods [Althoefer95e, Bugmann95, Lozano87], the fuzzy-based system is not plagued by the curse of dimensionality and can be applied to manipulators with a high degree of freedom (DOF). In the particular fuzzy implementation each link is controlled by a separate fuzzy unit. Any unit has three inputs at the most (see Section 4.3.2). Thus, the number of rules increases linearly with the degrees of freedom.

The suggested system is mainly meant to be implemented on-line. Nevertheless, it can also be implemented as an off-line planning strategy. In this case, the fuzzy navigator constructs a trajectory for a simulation model of the real manipulator in a simulated environment which represents the real workspace. The segments of the trajectory that is found can be then applied to the real manipulator. The advantage of this method is that when the method fails due to a dead-lock situation, other methods can be introduced to escape such a

[1] Distal links are links which are close to the end effector; proximal links are links which are close to the base.

situation without actually moving the real manipulator. Methods to escape a dead-lock, such as explorative random search, reversed start-goal location, etc., would produce undesirable manipulator movements if applied to the real arm. Compared to a global planner like the one presented in Chapter 3, the fuzzy navigator has a further advantage, namely it makes use of a workspace representation. Thus, the time expensive generation of a C-space representation is not necessary and in most situations the fuzzy controller would provide a solution faster.

The use of universal approximators, such as neural networks and fuzzy systems, to solve local navigation problems for mobile robots was investigated recently by many researchers (see for example [Reignier93, Tschichold96, Hoffmann96, Sharkey96]. These reactive or behaviour-based systems commonly incorporate two main types of behaviour: obstacle avoidance and goal directed behaviour. At different locations one property becomes more dominant than the other. For example the obstacle avoidance behaviour is strong when an obstacle is close, and weaker when no obstacle is in the vicinity of the robot. These two different behaviours compete with each other in the parallel processing systems and, thus, the final decision is based on multiple input signals. The rule base, which essentially defines the transfer function of the fuzzy system, differs from neural networks and analytical methods in that it is readable and can be easily adapted to the underlying problem.

The increasing interest in applying manipulators in safety critical areas (for example human support through robots, handling of expensive and fragile goods, etc.), has prompted an increasing demand for reliable and explainable control systems [Helliwell95, Morgan95, Redmill95]. To cope with specific safety requirements appropriate rules can be added to the rule base of a fuzzy controller.

It appears that fuzzy logic has not been previously applied to the obstacle avoidance problem for robotic manipulators. The novel conceptual approach which is proposed in this chapter will offer a new perspective to the field of collision avoidance and navigation in the area of robotic manipulators.

4.2 Local Navigation in Context

4.2.1 Artificial Potential Fields

In developing the presented technique, an obstacle avoidance method which is based on an artificial potential field was most influential [Khatib85, Khatib86, Reignier93][2]. Khatib made use of two artificial potential fields whose gradient produce forces which can be applied to a robot. One potential field produces a strong repelling force in the vicinity of obstacles while a second one produces an attracting force which influences the entire free space and reaches its maximum at the target location. The two independent fields are superimposed. The resulting force can be obtained by calculating the gradient of the superimposed fields. At any point in space, the direction of this force represents the most promising direction of motion [Latombe91].

[2] Note, this method is different from the resistive-grid approach presented in the previous chapter.

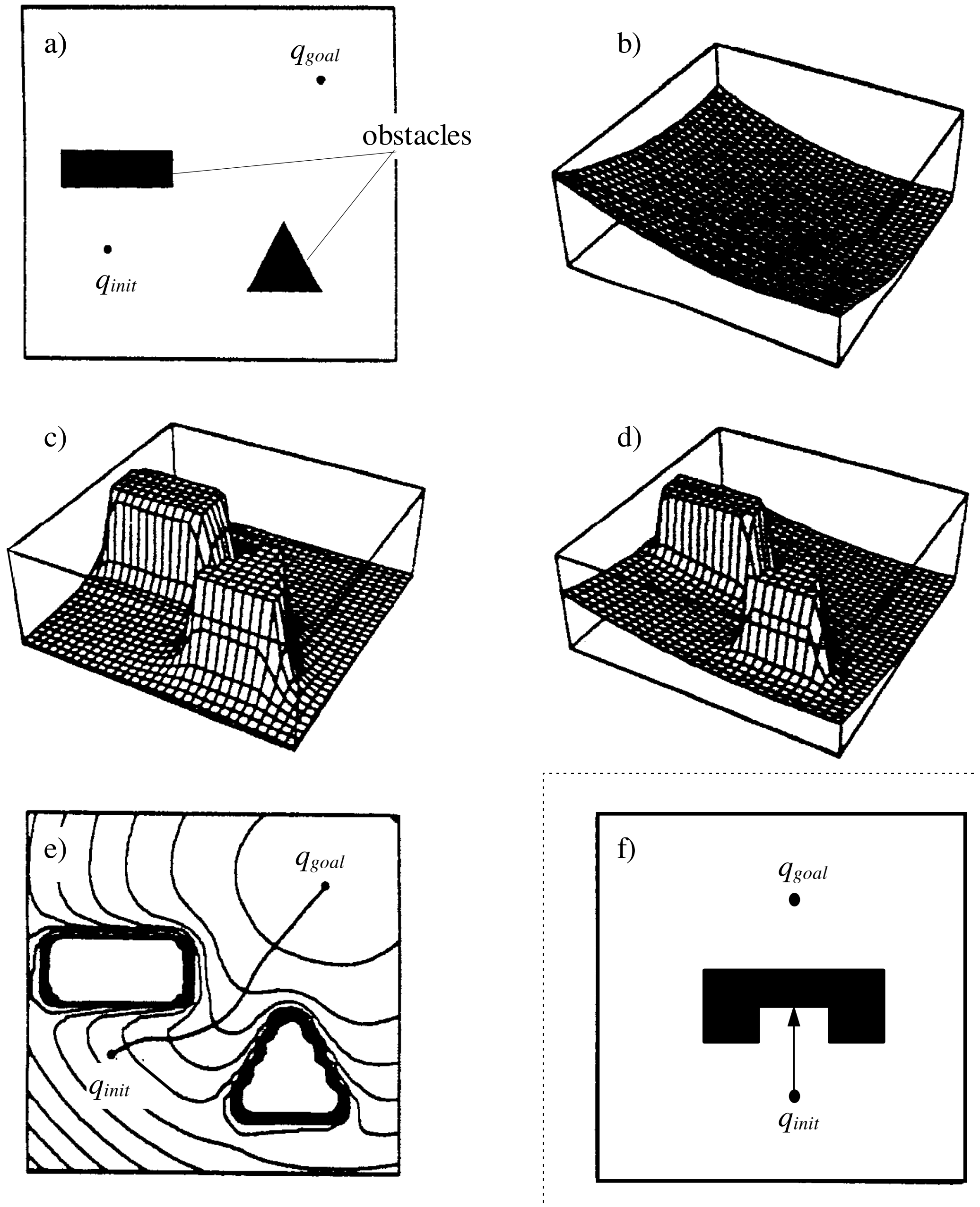


Figure 4.2-1: An example of Khatib's potential field method. Subfigure a) depicts the initial and the goal configuration of a robot in a configuration space with two obstacles. Subfigure b) and c) show the attractive and the repelling field, respectively. Subfigure d) shows the superposition of the two fields. The found path and the equi-potential lines of the potential field are depicted in subfigure e). Subfigure f) shows an example of a local-minimum (dead-lock) situation where the method fails. (after [Latombe91])

Most potential field methods in configuration space are based on the following equation:

$$\vec{F}(q) = -\vec{\nabla}U(q), \qquad (4.2\text{-}1)$$

where $F(q)$ represents the force at configuration q and $\vec{\nabla}U$ denotes the vector gradient of the potential field U at q [Latombe91]. The potential field U is the superposition of the two basic potential functions:

$$U(q) = U_{att}(q) + U_{rep}(q), \tag{4.2-2}$$

where the attracting function is given by U_{att} and associated with the target configuration, while the repelling one is given by U_{rep} and associated with the C-obstacle region. Thus, the resulting force is

$$F(q) = F_{att}(q) + F_{rep}(q) = -\nabla U_{att}(q) - \nabla U_{rep}(q), \tag{4.2-3}$$

where F_{att} and F_{rep} are denoted as the attracting and the repelling force, respectively.

Different potential functions are in use. The main purpose of the repelling force is to create a blocking shield around the C-obstacles which keeps the robot from crossing. While the attracting force is preferred to be effective in the entire C-free region, the repelling force should be restricted to the vicinity of the C-obstacle regions so that the robot particle is not affected when it keeps a sufficient distance (see [Khatib86, Latombe91]).

The above method is most easily applied to a small-sized robot in its workspace (see Figure 4.2-1) or a point-sized "C-space" robot in its configuration space. Nevertheless, the method has been also applied to multi-link manipulators in workspace. Then the potential fields are applied to so-called control points along the manipulator's links (for details see [Khatib86]).

The potential field method is especially useful when applied to a robot in its workspace because a time costly workspace to C-space transformation is thus avoided. However, when a C-space representation is available, it is probably less time-costly to plan a path employing a global planning strategy, such as the resistive grid (see Chapters 2 and 3). The potential field method might outperform the global planner if no local minimum occurs between start and goal but when the method is stuck in a local minimum (as for example shown in Figure 4.5-2) further strategies (e.g. random search) have to be employed to free the robot. In this case the potential field method might lose out against the global planner in terms of time. These comments apply not only to the potential field method, but to any local navigation technique.

4.2.2 An Overview of Fuzzy-Based Navigation Techniques for Mobile Robots

The application of fuzzy-based systems to robots has recently attracted a great deal of interest in those research communities which focus on the development of intelligent and autonomous mobile robots [Reignier93, Tschichold96, Roth93, Maier95, Song92, Maties94, Hoffmann96, Skubic93, Mitchell96]. Robots steered by commands from a fuzzy navigator have been also described as behaviour-based or reactive systems which are derived from the subsumption architecture proposed by Brooks [Brooks86].

The simplicity of the problem (a mobile robot is often represented by a single point), facilitates the conduct of experimental studies on a simulation basis. The problem of navigating a multi-link manipulator through an environment cluttered with obstacles is by far a more demanding task. However, the fuzzy-based system presented here shows similarities to systems utilised for the navigation and obstacle avoidance of mobile robots.

In [Kosko92], Kosko *et al.* describe the application of a fuzzy controller to reverse a truck to a predefined docking position. The output variable of the fuzzy controller is the steering angle of the truck. The inputs represent the truck angle and the truck position in respect to the desired landing point at the docking ramp. Kosko *et al.* compare their approach to the well-known neural-network-based approach of controlling a truck backer-upper by Widrow *et al.* [Nguyen90]. Kosko's simulation results show that the fuzzy controller is superior to the neural controller. The fuzzy controller used different rule bases; one was built up using common sense, while others were built employing different training methods (for details, see [Kosko92]). The fuzzy controller provided a smooth path in every experiment, while the neural controller, which was trained using the time consuming backpropagation algorithm, backed up the truck occasionally along an irregular path. Furthermore, the neural controller is reported to be computationally heavier than the fuzzy controller. While the neural network had to carry out multiplications, additions and logarithms, the fuzzy controller made do with comparisons and additions [Kosko92]. More importantly, the functioning of the fuzzy controller can be interpreted and specific adjustments can be made at any stage. The neural network's black box behaviour prevents the user from investigating the network's weight matrix, and changes of the weights can be only achieved by going through further training cycles.

Tschichold-Gürman developed a neural network architecture, called RuleNet, and an expansion on that, called Fuzzy RuleNet. The latter combines aspects of neural networks and fuzzy logic. Fuzzy RuleNet has properties similar to those of a fuzzy-based system, but its structure is similar to the one of a feedforward radial-basis-function network (see also Chapter 2). The hidden nodes of the Fuzzy RuleNet behave like the fuzzification stage in a fuzzy controller, while the output nodes provide the defuzzification. Like any feedforward neural network, the weights of the Fuzzy RuleNet can be adapted to the environment using a learning algorithm. The method suggested by Tschichold-Gürman has been tested in robot simulations as well as in real-world applications [Tschichold96, Vestli94]. Tschichold-Gürman *et al.* applied the Fuzzy RuleNet to different types of mobile robots, each one equipped with a computer system and sensors (such as laser rangefinder, ultra-sonic and infra-red sensors, and bumpers). To cope with different environments and changing tasks, they propose a global architecture which has in its core a situation-based behaviour selector. Depending on the sensory input and the required task, the selector chooses a behaviour module which is a neuro-fuzzy controller trained to react appropriately to the given circumstances [Tschichold96, Tschichold96b].

Other fuzzy-based navigators which provide a mobile robot with an actuation in reaction to the nearby environment can be found in literature. Examples of these methods can be found in [Reignier93, Roth93, Maier95, Song92, Mitchell96, Hoffmann96].

The fuzzy controller proposed in this chapter is based on the principles of the fuzzy controller developed by Mamdani and Sugeno [Mamdani81a, Sugeno84, Nauck94]. Recent advances with regard to fuzzy control in general as well as with regard to the training and tuning of fuzzy controllers can be used to further improve the proposed fuzzy navigator (see for example [Hoffmann96, Kosko92, Berenji92, Jang92, Reigner94]).

4.2.3 Unreachable Situations and Local Minima

The fuzzy navigator can get stuck in certain situations. In these dead-lock situations, the manipulator is not able to reach the desired goal configuration. These situations are similar to local minima in potential fields (see Figure 4.2-1). Nevertheless, it has been shown that if the state space to be searched is non-maze-like (as described in [Millan92]), elaborate path planning skills are not necessary and simple navigation techniques are in most cases successful [Millan92]. For the purposes of this chapter a non-maze-like workspace for a robotic manipulator is considered to be a space in which from any start to any goal configuration a path can be found that has a trajectory with parameters changing only in a monotonically increasing or decreasing manner. This assures that the manipulator will always reach its goal configuration. However, experiments showed that this restriction may be too strict, since more distal links do perform back-up movements when controlled by the proposed fuzzy navigator.

Different strategies have been suggested to overcome these dead-lock situations. One of them makes use of so-called via-points. These via-points can be provided for example by a human operator or a global planner [Tschichold96]. The path between two via-points is expected to be free of local minima. Other methods suggest the use of additional strategies, like random search, to escape those minima [Barraquand91, Latombe91]. However, the occurrence of these situations and local minima remains an issue, especially in the area of local navigation. An example of a such a situation which brings the navigator into a dead-lock situation is depicted in Figure 4.5-2.

4.3 Fuzzy Navigation and Obstacle Avoidance for Robotic Manipulators

4.3.1 Introduction to Fuzzy Control

Fuzzy logic is a well established method that has been already used to solve a number of control problems. It has been successfully applied to low-level motor control, the truck backer-upper problem [Kosko92, Higgins94], ship heading control [Brown94], behaviour based navigation for mobile robots [Tschichold96, Reignier93, Maier95, Song92], fuzzy-controlled object slipping by a manipulator gripper [Schmidt95], car traction control [Schuster94], communication systems [Kartalopoulos96], biotechnological processes [Serac96] as well as to the well-known inverted pendulum problem [Kosko92]. Fuzzy systems have also been implemented in a variety of industrial products. No indication could be found in the literature that suggests the use of fuzzy logic to solve the obstacle avoidance problem for robotic manipulators.

Fuzzy controllers are a means to control analogue processes. They are especially applicable when these processes are not easily described using conventional analytical tools, but can be described in form of rules based for example on the knowledge of a human expert. Obviously, humans are able to control many complicated systems without making explicit mathematical models of the system [Nauck94, Palm91b]. Thus, it is more sensible to model the behaviour of a human who controls a process than to model the process analytically as usually done in classical control. Observing the behaviour of human operators and interviewing them about their knowledge often allows the derivation of linguistic rules which can be used to form the rule base, also called the knowledge repository, of a fuzzy controller [Cox92]. Fuzzy controllers are characterised by a few parameters, impose a low computational burden, and produce multi-dimensional mappings from a set of input variables to one output variable or a set of output variables [Reignier93].

Fuzzy controllers are based on the fuzzy set theory proposed by Zadeh (see [Zadeh96, Kosko92, Droesser94]). Fuzzy set theory attempts to generalise the binary viewpoint which has been the base of the classical set theory [Kosko92]. Classical set theory says that an object is either member of a set or not. Fuzzy set theory generalises by proposing fuzzy sets. An object can belong to a fuzzy set to a certain degree. The degree of membership can vary between 0 and 1. Hence, an object with a membership degree of 1 belongs completely to a fuzzy set, while another object with a membership degree of 0 does not. In addition, all intermediate degrees of membership for an object belonging to a fuzzy set are possible. This theory allows the partitioning of a variable space into fuzzy sets. Any signal out of the variable space belongs more or less to at least one fuzzy set which can be described by a linguistic term. This procedure is called fuzzification (see also Section 4.3.3).

The principles of fuzzy control were for the first time described by Mamdani who developed a type of fuzzy controller named after him [Mamdani81a, Mamdani81b]. Many fuzzy controllers based on this controller type have been developed since (see for example [Kosko92, Sugeno84, Nauck94, Tschichold96]).

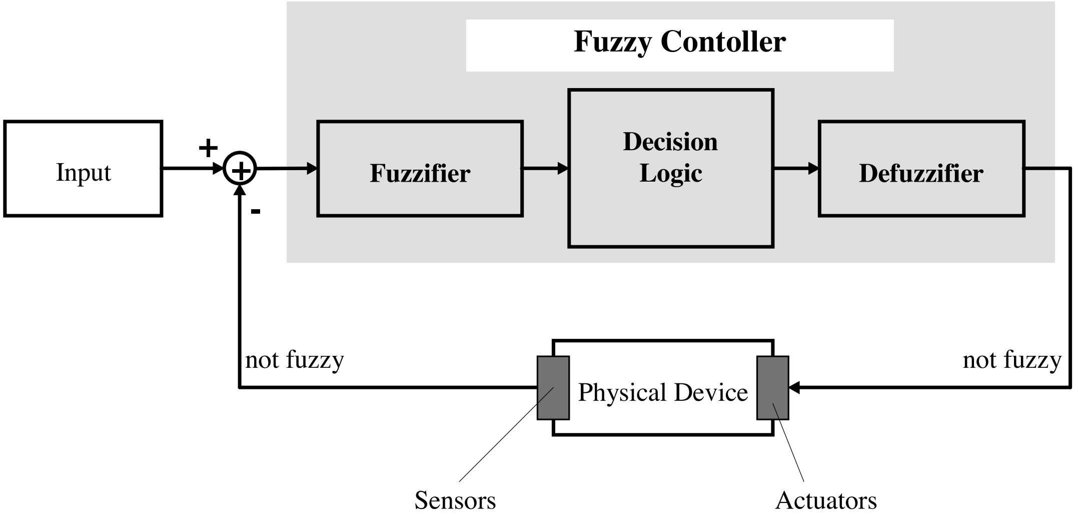


Figure 4.3-1: General depiction of a fuzzy control system.

The overall fuzzy control scheme is quite similar to a conventional control system, as it can be seen in Figure 4.3-1. This figure shows a feedback controller which steers a physical device, for example a robot. In contrast to classical controllers which usually make use of a mathematical algorithm, the fuzzy controller consists of three main sections: the *fuzzifier*, the *decision logic* and the *defuzzifier*. The controller considered in this thesis is based on the fuzzy controller suggested by Sugeno (see [Nauck94, Palm91b, Sugeno84]). In the Sugeno controller, the fuzzification of the output signal is omitted and the defuzzification stage is replaced by a weighted sum. Due to its computational efficiency, the Sugeno controller is commonly preferred to the Mamdani controller in real-time applications [Nauck94]. The remaining paragraphs of this section describe the functioning of this controller type. An illustrative description of the original Mamdani controller can be found for example in [Kosko92, Cox92].

The fuzzifier converts the input space, which usually describes the actual state of the device with reference to a desired state, into a fuzzy representation. In a robot navigation problem two input signals acquired by sensors are used to describe the device's state. The first signal is the difference between the actual state and the desired state of the device, ν , while the second input signal informs about the distance between links and obstacles, μ (see

Section 4.3.3 and [Althoefer96, Althoefer96c]). The fuzzifier partitions the space of an input signal into a number of fuzzy sets. In the case of the robot navigation problem, the input describing the difference between actual configuration and desired configuration is partitioned as follows: “*far left*”, ”*close left*”, ”*contact*”, “*close right*”, “*far right*” (see Figure 4.3-2), while the input describing the distance between robot and obstacle is partitioned as follows: “*far left*” , ”*left*” , ”*close left*”, “*close right*”, “*right*”, “*far right*”. Each of these fuzzy sets is sensitive to a range of input values and the degree of membership defines how much a particular signal value is part of a fuzzy set (see Section 4.3.3). The fuzzy sets are usually defined by functions which are non-zero in a small area only (for example triangular (Figure 4.3-2), trapezoidal or Gaussian functions are often used). The overlapping of different fuzzy sets allows the continuous approximation of input signals. This also means that one input signal can activate more than one fuzzy set. (Note, the similarity to the radial-basis-function network described in Chapter 2.)

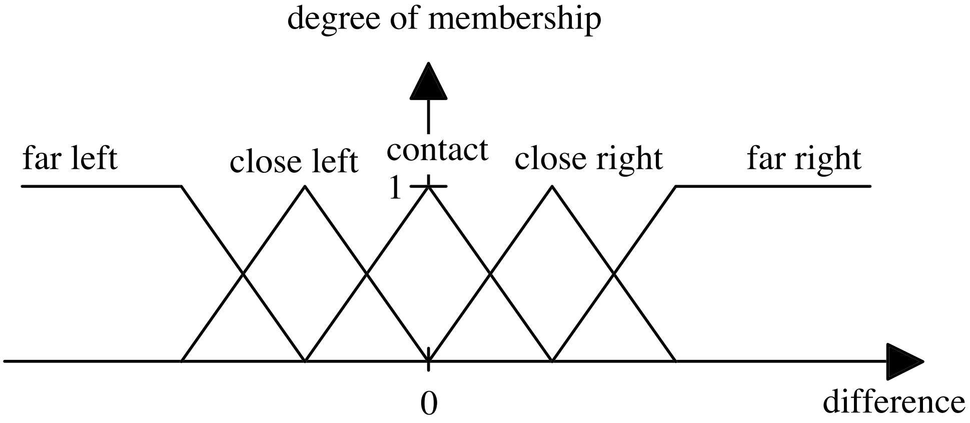


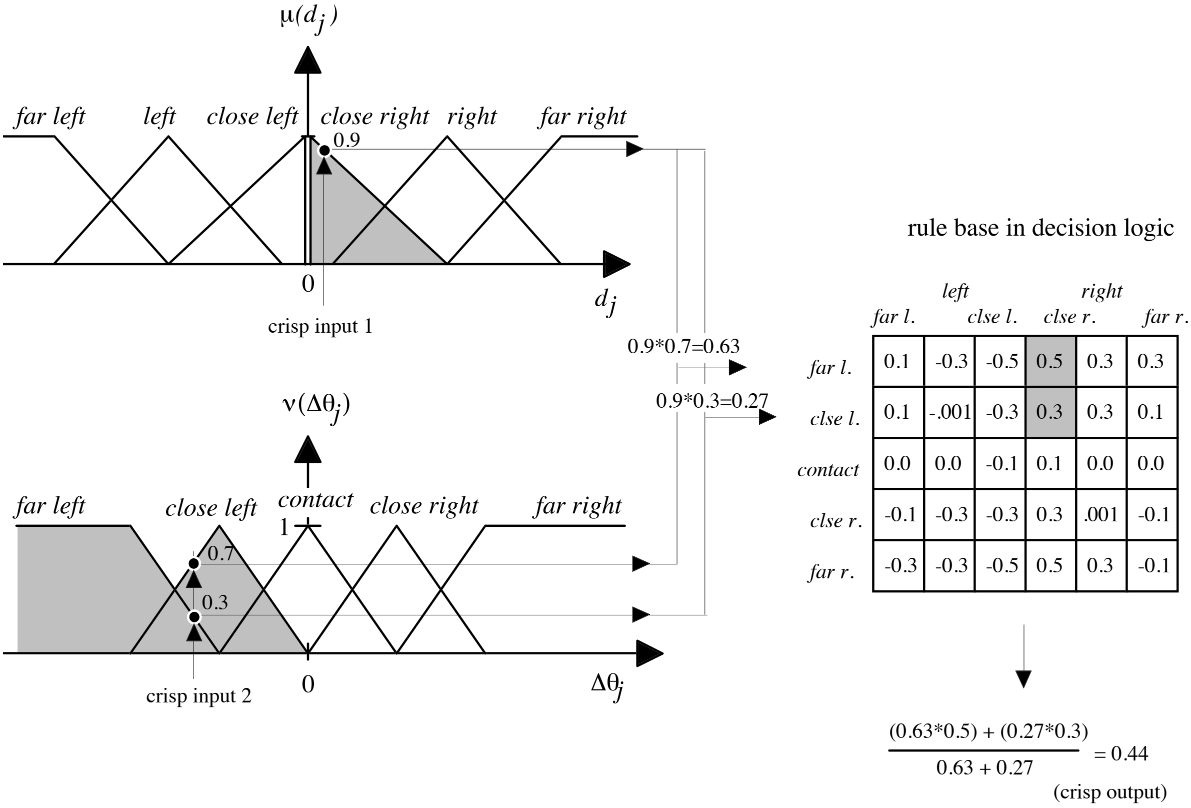
Figure 4.3-2: Five fuzzy sets partition one of the input spaces of the control system. The shown space represents the difference between actual and target configuration of the robot; in a similar way fuzzy sets can be set up to describe the distance between robot and obstacle (see also Figure 4.3-3).

μ ν	**far left**	**left**	**close left**	**close right**	**right**	**far right**
far left	left small	right big	right very big	left very big	left big	left big
close left	left small	right very small	right big	left big	left big	left small
contact	nil	nil	right small	left small	nil	nil
close right	right small	right big	right big	left big	left very small	right small
far right	right big	right big	right very big	left very big	left big	right small

Table 4.3-1: The fuzzy rule base for robot navigation. The two inputs, μ and ν, represent the fuzzy sets which describe the distance between robot and obstacle, and the difference between actual and goal configuration, respectively. The outputs of the base are the actions. The rule base defines a fuzzy association between outputs and paired inputs.

In a second stage, the output of the fuzzifier is presented to the decision logic. The decision logic comprises a rule base where outputs are associated with paired input fuzzy sets (see Figure 4.3-3 and Table 4.3-1). The rule base here is a two-dimensional look-up table whose elements are different scalar values representing actions, such as "*left big*", "*left small*", "*nil*", "*right small*", "*right big*", etc. Employing the robot navigation example once more, one rule might be: IF target is *far right* AND obstacle is *close left* THEN move *right big*. Thus, the decision logic generates a specific action which should be carried out on occurrence of certain signals at the inputs. The degree to which an input signal is member of a fuzzy set decides to what extent the connected rule or action is activated (see Section 4.3.3). As the fuzzy system is a parallel processing paradigm, more than one rule can be active at one time.

Finally, the multiple outputs of the decision logic (more than one cell in the look-up table of fuzzy rules may be active) have to be combined to achieve a crisp output signal [Kosko92, Droesser94]. In the Sugeno controller, the final output is the sum of the outputs of the decision logic weighted by the sum of the outputs of the fuzzifier (see also Section 4.3.3 and [Nauck94]). The final output signal usually drives an actuator which forces the device to change from its actual state into a new state.



		left			right	
	far l.		clse l.	clse r.		far r.
far l.	0.1	-0.3	-0.5	0.5	0.3	0.3
clse l.	0.1	-.001	-0.3	0.3	0.3	0.1
contact	0.0	0.0	-0.1	0.1	0.0	0.0
clse r.	-0.1	-0.3	-0.3	0.3	.001	-0.1
far r.	-0.3	-0.3	-0.5	0.5	0.3	-0.1

$$\frac{(0.63*0.5) + (0.27*0.3)}{0.63 + 0.27} = 0.44$$

Figure 4.3-3: Schematic depiction of fuzzification stage, decision logic and "defuzzification" of a fuzzy controller used for the robotic navigation task. The two inputs ("difference between actual and target position" (ν) and "distance to obstacle position" (μ)) are partitioned into a number of fuzzy sets. Two exemplary input values are indicated by "crisp input 1" and "crisp input 2". These input values activate the corresponding fuzzy sets (shaded fuzzy sets) which are then combined using the fuzzy product operator. In the next step, two actions in the two-dimensional rule base (shaded squares) are activated. These two actions are combined (weighted sum) to produce one crisp output. The crisp output represents an actuator command which lies between "*right very big*" (0.5) and "*right big*" (0.3). (Further explanations, see Section 4.3.3.)

4.3.2 Manipulator-Specific Implementation Aspects

This section focuses on the application of the suggested navigation technique to non-branching manipulators with revolute joints (see Figure 4.3-4). Those types of manipulators (for example PUMA series, SCARA design, etc.) are utilised in many industrial applications.

The technique consists of separate fuzzy-based obstacle avoidance units, each controlling one individual link, $l_j, j = 1,...,n$. Each unit has two main inputs: 1) the distance between the link and the nearest obstacle, d_j, and 2) the difference between the current link configuration and the target configuration, $\theta_j - \theta_{j,t\arg et} = \Delta\theta_j$. The output variable of a unit is the motor command, τ_j. All these variables can be positive as well as negative, thus, they do not only inform about the magnitude, but also about the sign of displacement relative to the link - left or right. The motor command which can be interpreted as an actuation for the link motor is fed to the manipulator at each iteration (see Figure 4.3-4).

For the calculation of the distance the only obstacles considered are those which fall into a bounded area surrounding each link and moving along with it. In this implementation, each such area is chosen to be of rectangular size. The area is as long as the link and reaches up to a predefined horizon on the left and right of the link (see Figure 4.3-4). This area can be seen as a simplified model for the space scanned by ranging sensors (for example ultra-sonic sensors) attached to both sides of a link [Risse95]. In case a three-dimensional manipulator has to be navigated in a three-dimensional environment, ranging sensors have to be mounted in such a way that they scan a cylindrical volume around each link. Of course, other shapes to describe the two-dimensional scan areas are conceivable. It is for example advisable to deal with the blind zones near the joints when the magnitude of the angles is large so as to assure that small moving obstacles are not missed by the algorithm [Risse95]. The wrong interpretation of the manipulator as an obstacle can be eliminated by examining the robot's kinematic [Risse95]. This problem is not considered here, since this chapter mainly deals with the investigating of the principles of the fuzzy-algorithm.

Besides an input from ultra-sonic sensors, a camera can be used to acquire the environment. Either a stationary camera or a set of cameras which oversee the entire workspace can be utilised (see Chapter 2 and [Jaitly96b, Ritter92]). Additionally or alternatively, a camera system which is mounted on the manipulator links can be used. If the manipulator is only equipped with a camera system that is attached to its links, it is important that the images provide a picture of the whole workspace. When only a look-ahead camera is attached, environmental changes behind the camera cannot be perceived and dealt with.

The task of a fuzzy unit is to provide a control function which produces an appropriate motor command from the given inputs. In broad lines, the control function can be described as follows: on the one hand the function has to lead the corresponding link to its attracting endposition; on the other hand it has to force the link to back up when approaching an obstacle which conveys a repelling influence. The fuzzy-rule-base (which represents the control function in a fuzzy unit) is built up by using common sense rules which are then refined by trial and error. In the case of the obstacle avoidance problem, the rule base has to evaluate the interaction of those competing inputs which are often contradictory in their character and produce an appropriate decision about the future action (motor command).

In this particular implementation, at each iteration the distance of the nearest obstacle on the left ($d_{j_{\text{left}}}$) and on the right ($d_{j_{\text{right}}}$) of link l_j are fed sequentially into the fuzzy unit (see Figure 4.3-4 and also [Hoffmann96, Risse95]). This process could be also carried out in a parallel fashion where two equivalent fuzzy controllers compute the response for the left and the right obstacle separately. The resulting two motor commands are superimposed, hence, both obstacles influence the final motor command which is applied to the link. The use of this method guarantees that the repulsion caused by one obstacle on one side of the link does not result in a collision with a nearby obstacle on the opposite side (see also Section 4.4.1, [Risse95 and references therein]). Only those obstacles are considered which are the nearest on the left and right.

In addition, fuzzy units of distal links communicate information about the distance to their nearest obstacles on the left and right to units of more proximal links (see Figure 4.3-4). Once sent to fuzzy units of more proximal links, this information can be used by the decision process of those units to slow down or even reverse the motion of the more proximal links. Without this propagation of information the control strategy might fail in situations where one obstacle is "known" only to a fuzzy unit of a distal link, while proximal links continue their motion based on their local environment dictating an adverse motion for the rest of the arm. This is especially important, since the same change of angle occurring at a proximal link and at a distal link produces a different velocity at the manipulator's tip. Thus, the motion of a proximal link might not be sufficiently compensated by an adverse motion at a more distal link.

Fuzzy units are only fed with the distance values of those obstacles which are inside the scan range. If no obstacle is detected inside a scan range, the fuzzy unit is informed of an obstacle which is *far left* or *far right*, respectively.

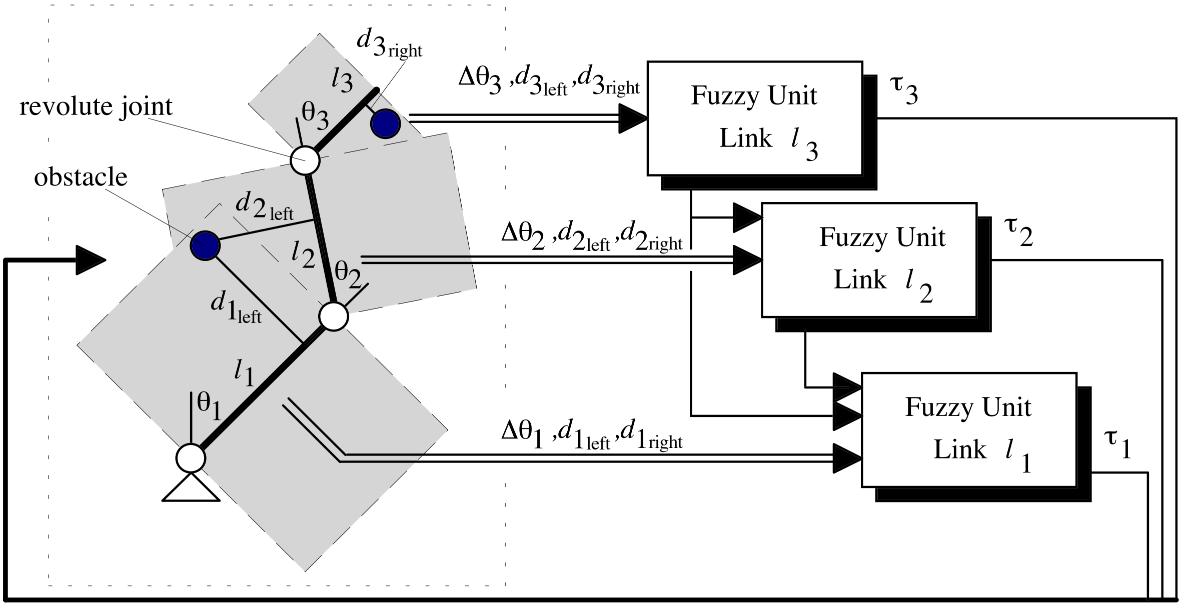


Figure 4.3-4: This figure shows a three-link planar robotic manipulator connected to fuzzy units. The shaded areas depict the regions which are "scanned" for the occurrence of obstacles. Each fuzzy unit receives via one input the difference between target and actual configuration, and via a second input, two values in a sequential way representing the distance between corresponding link and the nearest obstacle on the left and right of this link. If no obstacle is detected inside the scan area, the fuzzy unit is informed of an obstacle in the far distance. Additionally, proximal units are informed about obstacles in the vicinity of more distal links. Further explanations, see Figure 4.3-5, Sections 4.3.2 and 4.3.3.

Section 4.3.3 provides a general and theoretical treatment of the proposed fuzzy control system. This elementary approach assumes that for each newly added link the number of inputs of the more proximal fuzzy units increases by one. Assuming, that each fuzzy input is partitioned into five fuzzy sets, the most distal link has two inputs. (Via the first input, the unit receives in a sequential way the two values describing the link-to-right-obstacle distance and the link-to-left-obstacle distance, respectively. Via the second input, the unit receives the actual-to-target-configuration difference.) For a complete rule base, 5×5 = 25 rules would be need. The unit for the next link, $l_{n\text{-}1}$, has one further input (which describes the link-to-obstacle distances for the previous link, l_n) and its rule base would already consist of 5×5×5 = 125 rules. In a three-link system, the fuzzy unit responsible for the most proximal link would have a rule base with 5×5×5×5 = 625 rules, and so forth. This leads to a dimensionality problem. Hence, in the practical implementation presented in Section 4.4.2, the unit of a proximal link receives besides its two local inputs a third input which represents the smallest distance of all the more distal links to their obstacles on the left and on the right, respectively. In a sense, the fuzzy units of more proximal links are provided with a reduced information about the obstacle-to-link relation of the distal links. This approach allows the application of the fuzzy navigator to manipulators with a high degree of freedom. It limits the number of inputs per fuzzy unit to three, and, hence, the number of rules in the total system increases linearly with the number of links.

4.3.3 The Fuzzy Algorithm

While the previous section suggested a more practical implementation of the fuzzy navigator overcoming dimensionality problems, this section describes the navigator and its components in a more general and theoretical fashion.

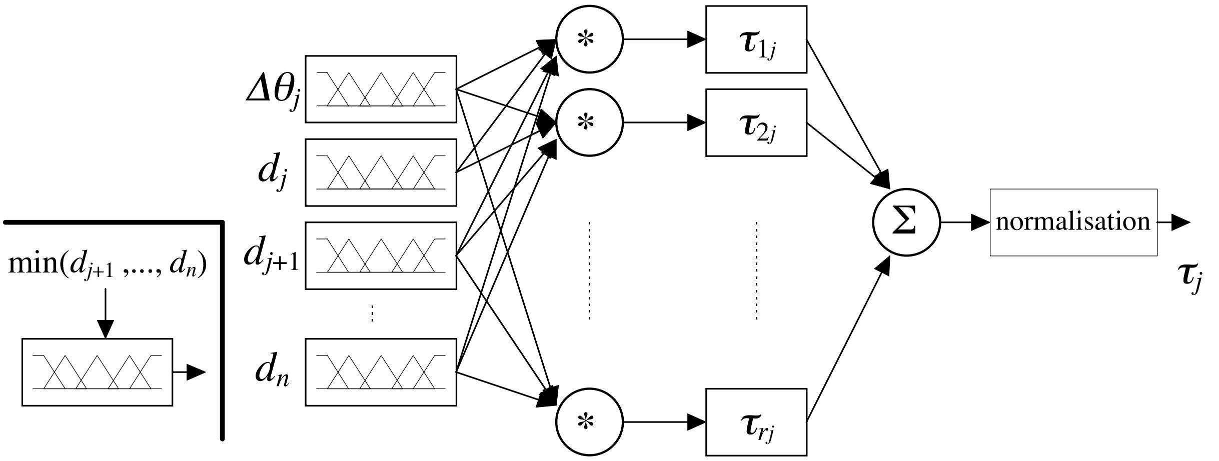


Figure 4.3-5: A more neural-network-like depiction of the controller used in the fuzzy unit of link l_j, $j = 1, ..., n$ (n is the number of links). The unit is influenced by obstacle-to-link distance values of link l_j as well as by obstacle-to-link distance values of links l_{j+1} ,..., l_n. In Section 4.3.2, the obstacle-to-link distance values of links l_{j+1} ,..., l_n were for practical reasons combined into a single value describing the minimum of these distances, as indicated on the left hand side of this figure.

The first input of each fuzzy unit is the difference between the actual angle and the target angle, $\theta_j\text{-}\theta_{j,\,target} = \Delta\theta_j \in \Theta_j$, $j = 1,...,n$ (n is the number of links). The value, $\Delta\theta_j$, is positive if the target is on the right, and negative if the target is on the left. The second input receives values describing the distance between link l_j and the nearest obstacles on the left and right in the "scanned" region, $d_j \in D_j$. An obstacle on the left produces a negative distance value, while an obstacle on the right produces a positive one. The single output is the motor command, $\tau_j \in \mathrm{T}_j$. A positive motor command moves the link to the left and a negative one to the right (see also [Althoefer96]).

Each universe of discourse D_j can be partitioned by fuzzy sets $\mu_1^{(j)},...,\mu_{p_j}^{(j)}$. Each of the sets $\mu_{\tilde{p}_j}^{(j)}$, $\tilde{p}_j = 1,...,p_j$ represents a mapping $\mu_{\tilde{p}_j}^{(j)}(d_j){:}D_j \to [0,1]$ by which d_j is associated with a number in the interval [0,1] indicating to what degree d_j is a member of the fuzzy set. Since d_j is a signed value, "*close_left*", for example, may be considered as a particular fuzzy value of the variable distance and each d_j is assigned a number $\mu_{close\_left}(d_j) \in [0,1]$ which indicates the extent to which that d_j is considered to be *close_left* [Mamdani81a]. In an equivalent way, fuzzy sets $v_1^{(j)},...,v_{q_j}^{(j)}$ can be defined over the universe of discourse Θ_j. In contrast to the Mamdani controller, the Sugeno controller [Nauck94] has an output set which is not partitioned into fuzzy sets (see also Figure 4.3-5). Thus, the rule conclusions merely consist of scalar actions $\tau_{\tilde{r}_j}$, $\tilde{r}_j = 1,\ldots,r_j$ (r_j is the number of rules in the fuzzy unit of link l_j).

There is a variety of functions employed to represent fuzzy sets. Here, asymmetrical triangular functions which allow a fast computation, essential under real-time conditions, are utilised to describe each fuzzy set $\mu_{\tilde{p}_j}^{(j)}$, $\tilde{p}_j = 1,...,p_j$ (see Figure 4.3-3 and [Althoefer96]). The parameters, $ml_{\tilde{p}_j}^{(j)}$, $mr_{\tilde{p}_j}^{(j)}$, which are the *x*-co-ordinates of the left and right zero crossing respectively, and $mc_{\tilde{p}_j}^{(j)}$, which describes the *x*-co-ordinate where the fuzzy set becomes one, define the triangular functions:

$$\mu_{\tilde{p}_j}^{(j)}(d_j) = \begin{cases} \min\left(\left(d_j - ml_{\tilde{p}_j}^{(j)}\right) \Big/ \left(mc_{\tilde{p}_j}^{(j)} - ml_{\tilde{p}_j}^{(j)}\right)\ ,\ 0\right) & \text{if } \ d_j \le mc_{\tilde{p}_j}^{(j)} \\ \min\left(\left(d_j - mr_{\tilde{p}_j}^{(j)}\right) \Big/ \left(mc_{\tilde{p}_j}^{(j)} - mr_{\tilde{p}_j}^{(j)}\right)\ ,\ 0\right) & \text{if } \ d_j > mc_{\tilde{p}_j}^{(j)} \end{cases} . \qquad (4.3\text{-}1)$$

As commonly done, the triangular functions are continued as constant values of magnitude 1 at the left and right side of the interval:

$$\mu_1^{(j)}(d_j) = \begin{cases} 1 & \text{if } \ d_j \le mc_1^{(j)} \\ \min\left(\left(d_j - mr_1^{(j)}\right) \Big/ \left(mc_1^{(j)} - mr_1^{(j)}\right)\ ,\ 0\right) & \text{if } \ d_j > mc_1^{(j)} \end{cases} \qquad (4.3\text{-}2)$$

and

$$\mu_{p_j}^{(j)}(d_j) = \begin{cases} \min\left(\left(d_j - ml_{p_j}^{(j)}\right) \Big/ \left(mc_{p_j}^{(j)} - ml_{p_j}^{(j)}\right)\ ,\ 0\right) & \text{if } \ d_j \le mc_{p_j}^{(j)} \\ 1 & \text{if } \ d_j > mc_{p_j}^{(j)} \end{cases} . \qquad (4.3\text{-}3)$$

Equivalently, the fuzzy sets can be defined for $\Delta\theta_j$. The fuzzy sets $\mu_1^{(j)},...,\mu_{p_j}^{(j)}$ and $v_1^{(j)},...,v_{q_j}^{(j)}$ as functions of the two inputs d_j and $\Delta\theta_j$ are depicted in Figure 4.3-3.

Additionally to the two inputs d_j and $\Delta\theta_j$, each fuzzy unit (apart from the most distal one) uses the distance fuzzy sets of more distal links $\mu^{(j+1)}_{\tilde{p}_{j+1}},...,\mu^{(n)}_{\tilde{p}_n}$ for decision making to assure that proximal links are slowed down, in case a more distal link is about to collide with an obstacle. Thus, each unit uses the following fuzzy sets: $\mu^{(k)}_{\tilde{p}_k}$, $k = j, j+1,...,n$ and $v^{(j)}_{\tilde{q}_j}$. Each of the fuzzy sets $\mu^{(k)}_{\tilde{p}_k}$ and $v^{(j)}_{\tilde{q}_j}$ are associated with linguistic terms $A^{(j)}_{\tilde{p}_j}$ and $B^{(j)}_{\tilde{q}_j}$ respectively. Thus, for link l_j the linguistic control rules $R^{(j)}_1,...,R^{(j)}_{r_j}$, which constitute the rule base, can be defined as:

$$R^{(j)}_{\tilde{r}_j}\text{: } \mathbf{IF}\ d_j\ \mathbf{is}\ A^{(j)}_{\tilde{p}_j}\ \mathbf{AND...AND}\ d_n\ \mathbf{is}\ A^{(n)}_{\tilde{p}_n}\ \mathbf{AND}\ \Delta\theta_j\ \mathbf{is}\ B^{(j)}_{\tilde{q}_j}\ \mathbf{THEN}\ \tau_{\tilde{r}_j}, \qquad (4.3\text{-}4)$$

where $\tilde{r}_j = 1,...,r_j$ and r_j is the number of rules for the fuzzy unit of link l_j and $\tau_{\tilde{r}_j}$ is a numerical entry in the rule base used in the defuzzification process (see Eq. 4.3-6). The most popular methods to calculate the fuzzy intersection (fuzzy-AND) are the *minimum* and *product* operators. If the *minimum* operator is used, the minimum of the inputs is chosen. If the *product* operator is chosen, inputs are multiplied with each other. While the result of the first approach contains only one piece of information, the second approach produces results which are influenced by all inputs [Brown94].

Here, the fuzzy intersection is calculated by using the product operator:

$$\begin{aligned}\sigma_{\tilde{r}_j} &= \mu^{(j)}_{\tilde{p}_j,\tilde{r}_j}(d_j) \cap .. \cap \mu^{(n)}_{\tilde{p}_n,\tilde{r}_n}(d_n) \cap v^{(j)}_{\tilde{q}_j,\tilde{r}_j}(\Delta\theta_j) \\ &\underset{\left(\substack{\textit{using the}\\ \textit{product operator}}\right)}{=} \mu^{(j)}_{\tilde{p}_j,\tilde{r}_j}(d_j) * ... * \mu^{(n)}_{\tilde{p}_n,\tilde{r}_n}(d_n) * v^{(j)}_{\tilde{q}_j,\tilde{r}_j}(\Delta\theta_j)\cdot\end{aligned} \qquad (4.3\text{-}5)$$

Finally, in the fuzzy controller here, the output of the unit is given by a weighted average over all rules (see Figure 4.3-5 and [Althoefer96, Nauck94]):

$$\tau_j = \frac{\sum_{\tilde{r}_j=1}^{r_j} \sigma_{\tilde{r}_j} \cdot \tau_{\tilde{r}_j}}{\sum_{\tilde{r}_j=1}^{r_j} \sigma_{\tilde{r}_j}}. \qquad (4.3\text{-}6)$$

Although the above control type is different from the Mamdani controller, the same results can be achieved employing a centroid defuzzification applied to a output fuzzy set with singletons, as for example done in [Schmidt95].

Changing the number of fuzzy sets, the parameters, $ml^{(j)}_{\tilde{p}_j}$, $mr^{(j)}_{\tilde{p}_j}$, $mc^{(j)}_{\tilde{p}_j}$, as well as the values of the weights $\tau_{\tilde{r}_j}$ are different ways of adapting the fuzzy-system to the underlying control problem. As an example, the control rules for the most distal link, l_n, of a manipulator are depicted in Table 4.3-2. A schematic depiction of the fuzzy control unit of link l_{n-1} is shown in Figure 4.3-6. The number of fuzzy sets which fuzzify d_j and $\Delta\theta_j$ are chosen to be six and five respectively. All other parameters have been refined by trial and error (see Sections 4.4.1 and 4.4.2).

ν_n \ μ_n	**far left**	**left**	**close left**	**close right**	**right**	**far right**
far left	left small	right big	right very big	left big	left big	left big
close left	left small	right very small	right very big	left big	left big	left small
contact	nil	nil	right small	left small	nil	nil
close right	right small	right big	right big	left very big	left very small	right small
far right	right big	right big	right big	left very big	left big	right small

Table 4.3-2: The fuzzy rule base for the most distal link l_n. It has two inputs, μ_n and ν_n, and a corresponding weight value $\tau_{\tilde{r}_j}$ for each input pair. The experiments showed that the performance of the system is very robust against minor changes of most of the weights. The depicted rule base has been used to control link l_2 in the two-link-manipulator experiment shown in Figure 4.4-1 and Figure 4.4-2.

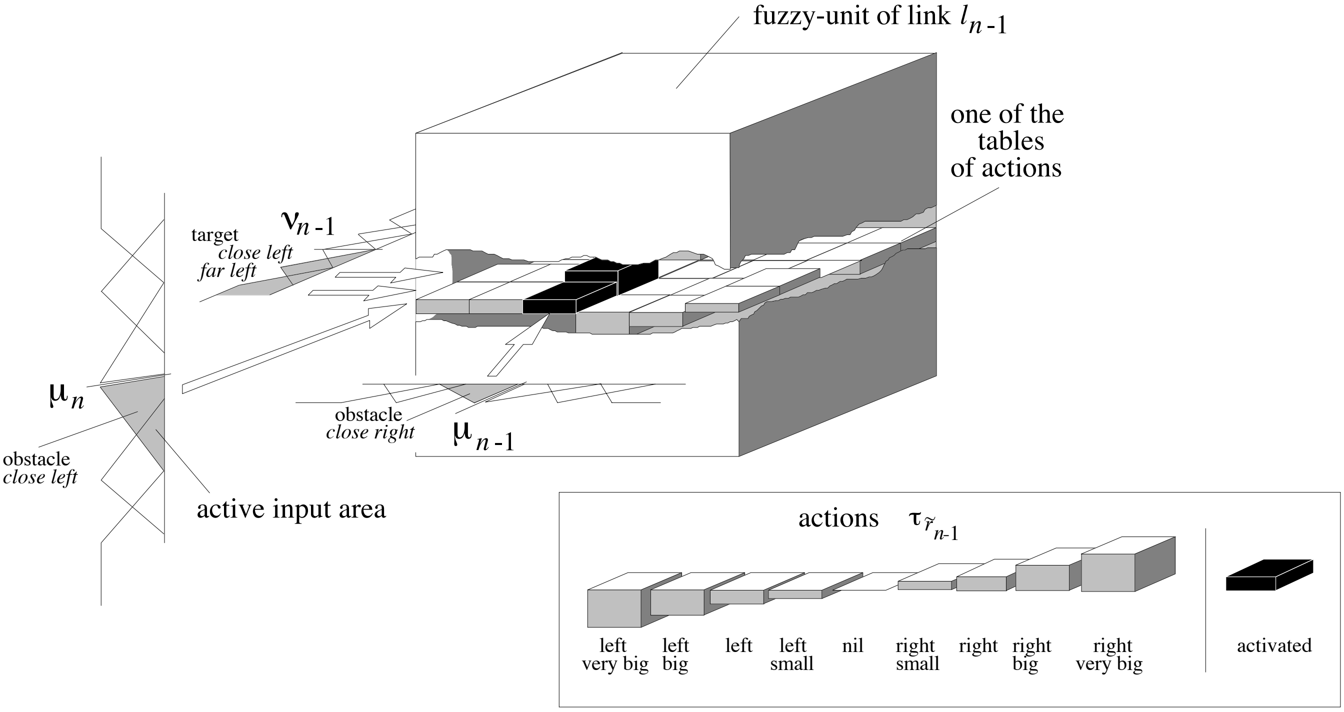


Figure 4.3-6: Schematic depiction of the fuzzy unit for link l_{n-1}. The fuzzy sets μ_{n-1} and ν_{n-1} describe the attracting and repelling "force" on link l_{n-1}, respectively. The fuzzy sets μ_n represent the influence of the more distal link l_n reflecting its distance to nearby obstacles. The overall motor command τ_{n-1} is a combination of the activated actions $\tau_{\tilde{r}\,n-1}$ which are represented by small black boxes in the table of actions. In this three-dimensional look-up table, there are as many two-dimensional tables of actions as there are fuzzy sets μ_n; only one of those tables is shown.

The performance of the fuzzy units can be studied by investigating their transfer functions. The transfer function of a fuzzy controller in the fuzzy-unit for the second link is depicted in Figure 4.3-7. Figure 4.3-7 shows the controller's output versus varying signals at the two inputs. The wing-shaped pattern represents the controller output for different obstacle-to-link distances for obstacles which are on the right side of the link. The transfer function with respect to an obstacle on the left (not shown) has a centre of inversion symmetry relationship to the transfer function generated by an obstacle on the right. Note that in most of the experiments conducted here the output of one fuzzy controller is superimposed onto the output of a second fuzzy controller which deals with the nearest obstacle on the opposite side of the link (see Figure 4.3-7 (right)).

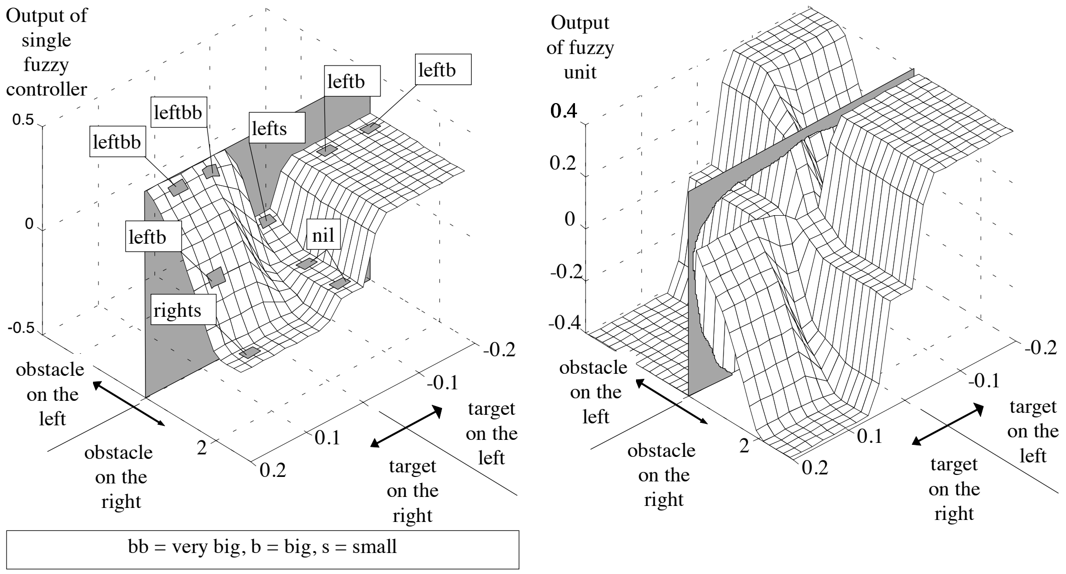


Figure 4.3-7: (left) The transfer function of a fuzzy controller in the fuzzy unit of link l_2. The patches denote the action values of the rule base. (right) The transfer function of the fuzzy unit of link l_2 where both fuzzy controller outputs are superimposed. One of these controllers suggests an action to be taken in response to an obstacle on the right, while the other suggests an action to be taken in response to an obstacle on the left. The grey partition separates the input space for obstacles on the right from the input space for obstacles on the left.

To generate the depiction of the transfer function in Figure 4.3-7 (left), the distances to the obstacle on the right are varied over the range of the input space. The grey patches indicate the area where certain rules are active. The linguistic terms which are written next to the patches are taken from the right part of Table 4.3-2. Three areas of the wing-shaped pattern describing the fuzzy controller's output can be well distinguished (Figure 4.3-7 (left)):

- (right part of the transfer function) It can be seen that the controller is constantly providing a positive output which represents an actuator command "*move to the left*" if the target is "*left*" or "*far left*". This response is independent of the distance to the obstacle on the right. Such a response is anticipated since a movement to the left is not hampered by an obstacle on the right.
- (middle part of the transfer function) Near the target the output is more or less zero. An obstacle which is "*close right*" provides a small repelling influence.
- (left part of the transfer function) When the target is on the right side of the link, the

obstacle avoidance behaviour of the fuzzy controller becomes obvious. Even though the target is on the right and the obstacle is "*close right*" or "*right*", the controller advises a movement to the left. Only when the obstacle is "*far right*", the controller allows "*right small*" movements. In this way it is assured that obstacles are avoided, and a target which is on the same side like an obstacle will be only reached if this obstacle is far away.

Similar arguments apply to the transfer function produced by an obstacle on the left. The investigation of these graphs can be used to show the performance of the controller and allow the designer of the fuzzy controller to find errors and conduct further refinement.

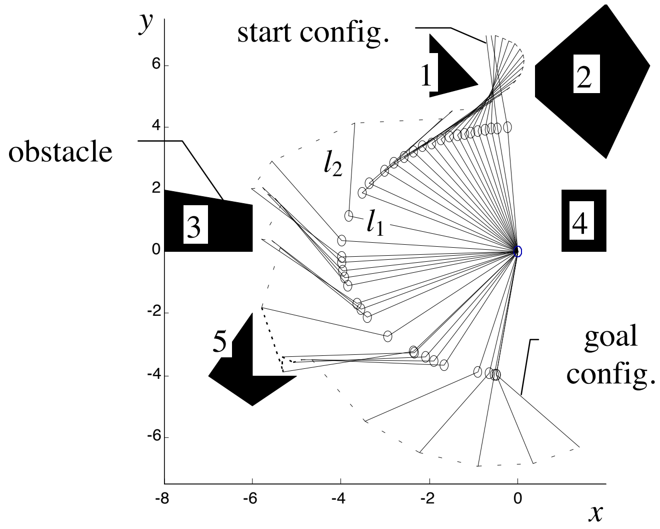
The transfer function of a fuzzy unit (depicted in Figure 4.3-7 (right)) is the result of the superimposition of the transfer functions of two fuzzy calculations - one dealing with the nearest obstacle on the right and the other dealing with nearest obstacle on the left. To allow a graphical representation, the output of the fuzzy unit can depend only on two inputs (here: target-to-actual-configuration difference and only one obstacle-to-link distance), thus, it was assumed that there is only one obstacle on one side of the link and no obstacle on the opposite side. This situation occurs when there are only obstacles on one side of the link in the scanned area surrounding the link. However, even in this case, both fuzzy controllers are active, since the fuzzy calculation which deals with the side where no obstacle is situated, generates signals according to the rules for obstacles which are "*far right*" and "*far left*", respectively (see Table 4.3-2). Although Figure 4.3-7 describes a special case, further experiments have been carried out where one obstacle changed its distance, while another on the other side of the link kept constant distance to the link. The resulting diagrams were very similar to Figure 4.3-7.

4.4 Computer Simulations

4.4.1 Two-Link Manipulator

The method as described in Section 4.3 has been applied to a two-link planar manipulator with stick-like links (see Figure 4.4-1, Figure 4.4-2 and Figure 4.4-3). The manipulator has a total length of seven units where link l_1 is 4 units long and link l_2 is 3 units long. The ratio of the two links is 4/3. The links of the robot arm have zero width. In most of the experiments, the rule bases have been set up using common sense and refined by trial and error.

Obstacles are represented by the union of discrete obstacle points along the obstacle's perimeters (see also Chapter 2). Each obstacle point is described by its xy-co-ordinates. In the examples, the obstacles are represented by the corner points only. In Chapter 2, it was shown how an obstacle image taken by a camera can be appropriately processed by some image techniques to produce discrete bitmap-like representation of the obstacle's perimeter. This approach could be also employed here to achieve an input for the fuzzy-navigator. The processing time of the fuzzy controller does not strongly increase with an increasing number of obstacle points, since after finding the closest points on the right and on the left, the subsequent calculations only consider these two points. Determining which are the closest obstacle points takes only a fraction of time compared to the calculation in the fuzzy controller.



rule base for link l_2

μ_2 / ν_2	*far left*	*left*	*clse left*	*clse right*	*right*	*far right*
far left	0.100	-0.300	-0.500	0.300	0.300	0.300
left	0.100	-0.001	-0.500	0.300	0.300	0.100
contact	0.000	0.000	-0.100	0.100	0.000	0.000
right	-0.100	-0.300	-0.300	0.500	0.001	-0.100
far right	-0.300	-0.300	-0.300	0.500	0.300	-0.100

Figure 4.4-1: A two-link manipulator controlled by the fuzzy-based strategy. In this case, the rule base was only coarsely designed; this resulted in an aggressive motion performance where link l_2 tries to reach its goal configuration quickly. The continuing tendency of link l_2 to reach its goal configuration and its backing-up near obstacles is obvious.

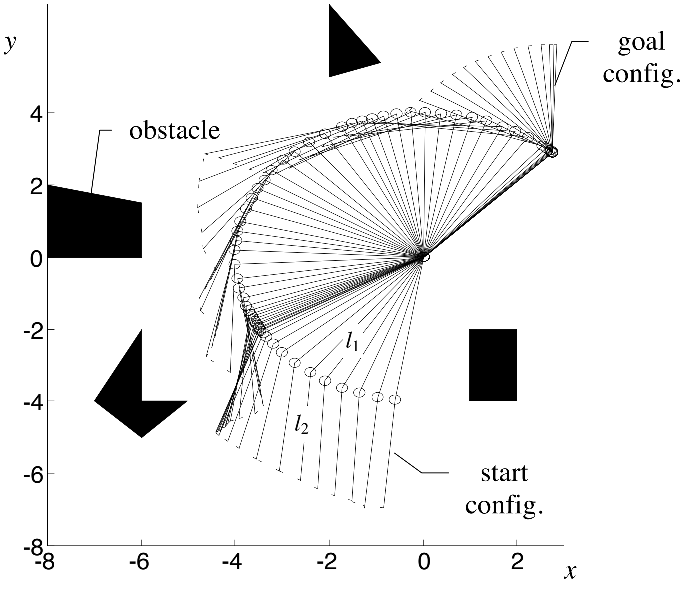


Figure 4.4-2: The trajectory depicted here was found using the same rule base as in Figure 4.4-1. This shows that a fuzzy controller designed for and improved in a particular environment is also able to cope with new environments and new start and goal configurations.

One of the experiments carried out using the fuzzy-navigator is shown in Figure 4.4-1. The linguistic terms used are depicted in Table 4.3-2 and the corresponding numerical values are shown at the bottom of Figure 4.4-1. The figure shows how the two-link manipulator moves from the given start to the given goal configuration. Obstacles are avoided successfully. Figure 4.4-1 also clearly shows the influence more distal links have on the more proximal ones. Each time, link l_2 leaves the influence of an obstacle, not only link l_2 but also link l_1 makes bigger forward steps. Obviously, the emphasis of investigation was on situations which result in a solution to be found by the proposed fuzzy navigator. Here, it was especially the intention of the author to show that those solutions are paths which are collision-free. In most of the experiments employing a well-adjusted rule base, this was the case. However, other experiments have been carried out where obstacles were in the range of link l_1 and no path connecting start and goal existed. In those cases, the arm kept trying to reach the goal configuration, but the repelling influence of the obstacle in the way prevented this, and the arm started slightly oscillating in a safe distance to the obstacles (see also Figure 4.5-2).

Figure 4.4-2 reveals the generalisation capability of the fuzzy navigator. Even though the same rule base as in Figure 4.4-1 has been employed, the navigator manages to move the manipulator in a different environment without causing a collision.

All experiments described in this chapter were carried out on an IBM-compatible PC with a PENTIUM90. To construct the above path, the fuzzy-algorithm run under MATLAB needed approximately 9.5 seconds. The path consists of 39 single steps. Thus, each iteration took in average 0.24 seconds if run under MATLAB. In view of the considerations made in Chapter 3 and announcements by [Matlab96], this value might be considerably reduced, if the program is run in compiled form (reduction by a factor 100 to 200 is achievable). Hence, the response time of the fuzzy controller might be reduced to about 1.2 or 2.4 *ms*. This makes the fuzzy controller suitable for real-time applications.

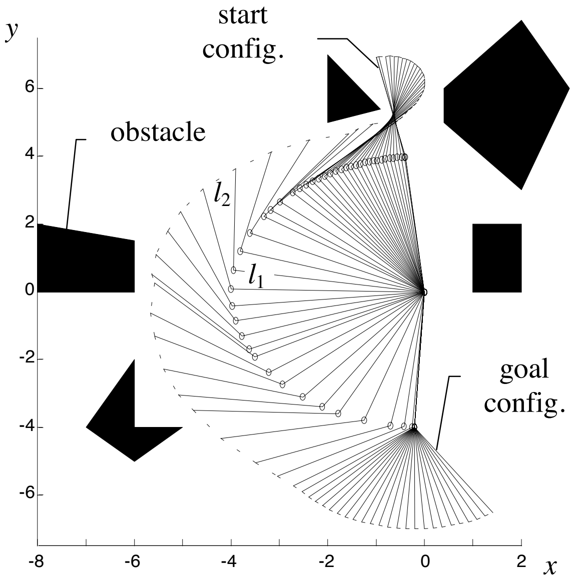
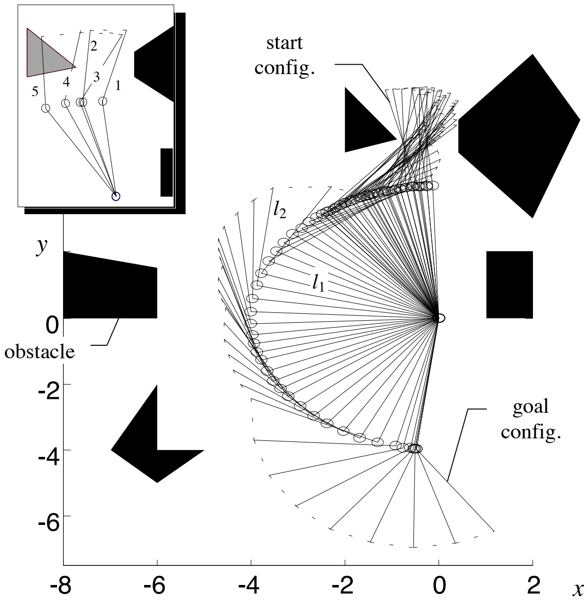


Figure 4.4-3: The two-link manipulator with a more refined rule base.

The fuzzy-navigator used for the experiment shown in Figure 4.4-1 makes use of an "aggressive" rule base which attempts to "push" link l_2 towards the final position as quickly as possible. This means that in obstacle-free areas, the links make big steps towards the goal configuration. A less aggressive rule base was used for the experiment shown in Figure 4.4-3. The rule base of this experiment has smaller action values for those rules which are active when target and obstacles are far. On the one hand, this rule base produces a smoother path and keeps in general a greater distance to obstacles; on the other hand, more steps are needed to reach the goal configuration, as it can be seen especially at the end of the path where link l_2 takes a lot of small steps to reach its goal configuration.



rule base for link l_2

μ_2 / v_2	*far left*	*left*	*clse left*	*clse right*	*right*	*far right*
far left	0.010	-0.100	-0.100	0.100	0.100	0.300
left	0.010	-0.001	-0.100	0.100	0.100	0.300
contact	0.000	0.000	-0.001	0.001	0.000	0.000
right	-0.300	-0.100	-0.010	0.100	0.001	-0.010
far right	-0.300	-0.100	-0.010	0.100	0.100	-0.010

Figure 4.4-4: In contrast to previous experiments where the influence of the nearest obstacle on the left and the nearest obstacle on the right were superimposed only the nearest obstacle is considered here. This approach tends to produce strong oscillations, if a link is situated between two obstacles (compare to Figure 4.4-1). The oscillation can become so strong that collisions with obstacles occur as shown in the inset where link l_2 enters the shaded obstacle. Numbers in the inset denote the order of movements. The rule base for link l_2 is quite different from the one shown in Figure 4.4-1. This is mainly because in the earlier experiments, the final output of a fuzzy unit was influenced by obstacles on both sides. Too strong actions in the rule base here may result in a collision, since there is no compensation from the other side. The rule base for link l_1 needed only minor modifications.

In the above experiments, the influence of the obstacle which is located on the right of a link as well as the one on the left have been superimposed to calculate the final actuator command for the link. In the next experiment only the influence of one obstacle or, to put it better, one obstacle point has been considered. At each iteration, the distances to all obstacles in the scan area of a link (Figure 4.3-4) are calculated. Only the one with the smallest value is used as input to the fuzzy-unit, thus, no superimposition is carried out.

This approach works successfully in those cases where obstacles are only on one side of the link. In cases where the link tries to navigate between two obstacles, the method can produce oscillations as depicted in Figure 4.4-4. These findings have been also reported by Maciejewski *et al.* (see [Risse95 and references therein]). The oscillations are caused by the repulsion of the link by the nearest obstacle into the vicinity of the opposite obstacle which then becomes the nearest one and repels the link again towards the first obstacle. This process is repeated until the navigator manages to move the manipulator out of the gap. Depending on the size of the gap, the manipulator start configuration and, especially, the rule base, this oscillation can mount up and a link can be driven into an obstacle (shown in the inset of Figure 4.4-4). The superimposition of the left and right obstacle's fuzzy-controller output (as described in Section 4.3.2) reduces the strength of oscillations, since both obstacles contribute to the actuator command.

In this case, the fuzzy-algorithm run under MATLAB needed approximately 12.5 seconds to move the manipulator from start to goal configuration. The path consists of 77 single steps. Thus, each iteration took in average 0.16 seconds. Although the response time is shorter than the one of the above discussed experiments, there is a great amount of oscillations at the beginning of the path, and the controller needs a lot of small adjustments to move the arm out of the gap between the two upper obstacles. This results in an overall longer time for the entire path.

4.4.2 Three-Link Manipulator

In this section, the fuzzy-based algorithm has been applied to a three-link manipulator. The three-link arm has a total length of 8.5 units where link l_1 is 4 units long, link l_2 is 2.5 units long and link l_3 is 2 units long. Again, links of the robot arm have zero width. The algorithm makes use of the superimposition of the influence of the left and right obstacle. The rule bases have been set up using common sense and refined by trial and error.

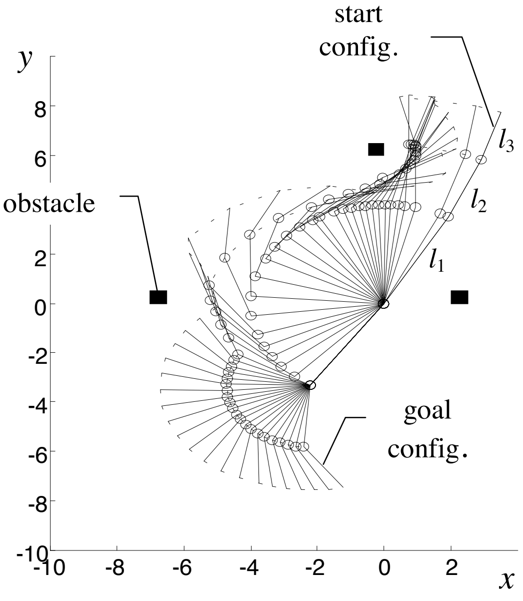
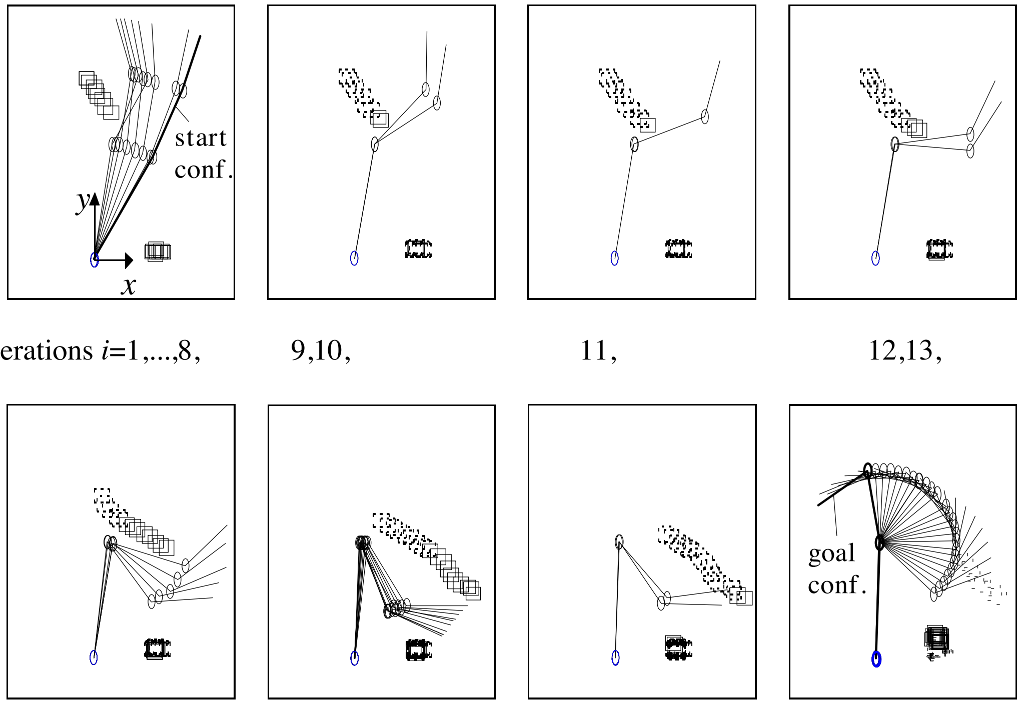


Figure 4.4-5: The application of the fuzzy-based strategy to a three-link manipulator.

To construct the above path, the fuzzy-algorithm which run under MATLAB needed approximately 23 seconds. The path consists of 49 single steps. Thus, each iteration took in average 0.47 seconds if run under MATLAB. Assuming a reduction by a factor 100 to 200, the response time of the fuzzy controller might be reduced to about 2.3 or 4.7 *ms*. These values achieved for a three-link manipulator are still good enough for real-time applications.

As only links with zero inertia are considered in all the above simulations, the applied motion command is an instantaneous angular step which modifies the previous velocity without time delay. Nevertheless, it has been shown that the technique can also be applied to a physical manipulator by appropriately adjusting the parameters to cope with the manipulator's dynamics (see also Section 4.5.2). To improve the performance of the system, higher order signals (velocity, acceleration) can be incorporated in the rule base.

The strategy has shown a very robust and stable performance. It produced in most experiments a collision-free path although these experiments were conducted in different simulation workspace scenarios. An appropriate rule-base could be easily designed, even without detailed knowledge of the particular obstacle avoidance problem.



Iterations i=1,...,8, 9,10, 11, 12,13,

14,...,18, 19,...,28, 29,30, 31,...,56.

Figure 4.4-6: This figure shows a three-link manipulator which moves from a start configuration to a goal configuration avoiding moving obstacles. To show that the manipulator successfully "out-manoeuvres" the obstacles, the sequence of motion has been depicted in a series of figures. Squares represent obstacles; past obstacles are depicted with dotted lines The upper obstacle performs a linear motion on which a random movement is superimposed. The lower obstacle performs a purely random motion (see also [Althoefer96c]).

4.4.3 Moving Obstacles

The described algorithm has been also applied to the three-link manipulator in dynamic environments (see Figure 4.4-6 and [Althoefer96c]). Even though the rule base was constructed during a trial-and-error procedure in a static environment, the proposed fuzzy algorithm behaved correctly when confronted with moving obstacles. The sequences of motion depicted in the subfigures of Figure 4.4-6 demonstrate that at each iteration *i* the manipulator attempts to reach the target configuration, but is repelled by any obstacle in the way. It can be seen that the moving obstacle even forces the manipulator to move backwards (see Figure 4.4-6).

4.4.4 Safety Aspects

In applications where the manipulator interacts with humans or expensive equipment, safety constraints have to be imposed to prevent damage. In those cases a fuzzy approach, as suggested here, is suitable because in contrast to neural networks, the rule-based fuzzy approach convinces with its transparency and allows the designer to shape the decision process by altering rules individually so as to achieve a specific performance. Although, a well-tuned fuzzy-navigator produced in most environments a collision-free path, it cannot be guaranteed that no collision will occur in any situation. To improve the safety of the navigator, one can either add a mechanism which is on a higher hierarchical level and decides to take on appropriate measures to deal with a safety-critical situation. Alternatively, certain "dangerous" situations can be individually described and an appropriate motor command can be integrated into the navigator's the rule bases (Table 4.4-1). It may be necessary to appropriately adjust the defuzzification, to make sure that the safety commands are effective at all times and not overruled by other rules.

$\nu_n \setminus \mu_n$	far left	left	close left	**collision**	close right	right	far right
far left	left small	right big	right very big	**STOP**	left big	left big	left big
close left	left small	right very small	right very big	**STOP**	left big	left big	left big
contact	nil	nil	right small	**STOP**	left small	nil	nil
close right	right big	right big	right big	**STOP**	left very big	left very small	right small
far right	right big	right big	right big	**STOP**	left very big	left big	right small

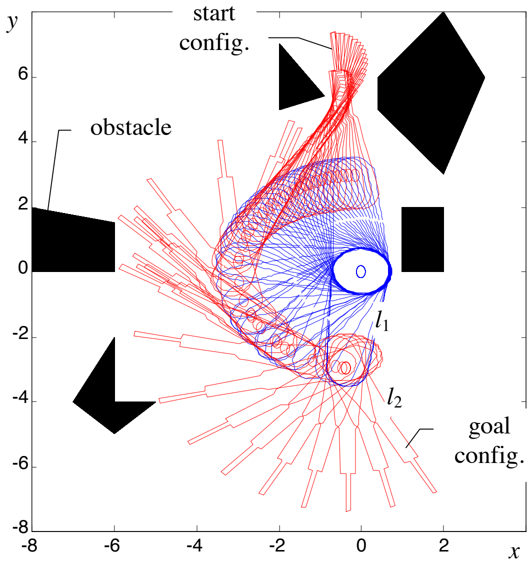
Table 4.4-1: The fuzzy rule base for a two-link manipulator. An extra column is added to Table 4.3-2. This column is active, if a collision occurs. A specific command can be invoked to deal with this safety-critical situation. In this example, a collision would lead to a immediate shut down of the manipulator links. The detection of such a safety-critical situation could be performed by special sensors, like bumpers for example.

Furthermore, it has been shown that in man-machine applications, as opposed to industrial ones, precision requirements are commonly lower than the demand for simplicity and a fast response time (see also [Morasso95, Zicky95]). Since the presented algorithm has a short feedback loop, a quick reaction to a changing environment is provided.

4.5 Fuzzy Navigation for the MA 2000 Manipulator

4.5.1 Simulated MA 2000 and Comparison to the Resistive Grid Approach

Prior to the real-world experiments (described in Section 4.5.2), the fuzzy-based navigation algorithm has been applied to the simulated version of the MA 2000-manipulator. This simulated version of the MA 2000 has two links, l_1 and l_2, which represent the shoulder and elbow link of the MA 2000, respectively. Link l_1 is connected via a revolute joint to a fixed base and link l_2 is connected via a second revolute joint to link l_1. The axes of the two joints are parallel. The vector model of the MA 2000 has been appropriately scaled to fit into simulation environments used in previous sections.



rule base for link l_2

μ_2 / v_2	*far left*	*left*	*clse left*	*clse right*	*right*	*far right*
far left	0.100	-0.150	-0.300	0.150	0.150	0.150
left	0.100	-0.001	-0.300	0.150	0.150	0.150
contact	0.000	0.000	-0.100	0.100	0.000	0.000
right	-0.150	-0.150	-0.150	0.300	0.001	-0.100
far right	-0.150	-0.150	-0.150	0.300	0.150	-0.100

Figure 4.5-1: The MA2000-manipulator steered by the fuzzy navigator. The manipulator has been appropriately scaled in size (keeping the proportions of the manipulator) to fit into the previously used simulation environment. Only slight changes were necessary to adapt the rule base of the stick-like arm to control the MA 2000 (compare tables in this figure and in Figure 4.4-1). The fuzzy unit for link l_1 has been taken over without changes.

The experiment here demonstrates the robustness of the developed algorithm. Only minor changes were necessary to adapt the fuzzy base, which had been set up for the stick-like arm (Section 4.4.1), to the MA 2000-shaped arm. This is remarkable considering the fact that the MA2000-manipulator has a greater link width and a different ratio between the links l_1 and link l_2 when compared to the stick-like arm. In fact, the fuzzy sets for the input space have been taken over without any change. The rule base for link l_1 has been also taken over without any change. The principal structure of the rule base for the second link remained also unchanged; some adjustments to the action values had to be applied. Thus, the rule base for link l_2 is still the same as shown in Table 4.3-2. The difference lies in the different values chosen to define the actions in the rule base. In general, the action values for the control of the MA 2000 are lower than those chosen for the stick-like arm (compare the tables in Figure 4.4-1 and Figure 4.5-1). The fuzzy-algorithm run under MATLAB needed 9 seconds, to construct the path in Figure 4.5-1. Since the path consists of 44 single steps, each iteration takes in average 0.2 seconds to run under MATLAB. The response time of the fuzzy controller might be reduced to about 1 *ms* if run as a compiled program. This is virtually the same value found for the two-link stick-like manipulator in Section 4.4.1.

The experiment depicted in Figure 4.5-1 is only a simulation where the manipulator instantaneously changes position. Thus, the given time of 9 seconds does not include the travel time of a real manipulator but is merely a sum of response times. If the fuzzy system were to be applied to the real arm, the time to traverse the whole path would alter and probably increase because the output of the fuzzy system depends on the feedback signals from the sensors and, therefore, depends on the properties of the particular manipulator (inertia of the links, actuator torque, friction in the gearboxes of the joints etc.).

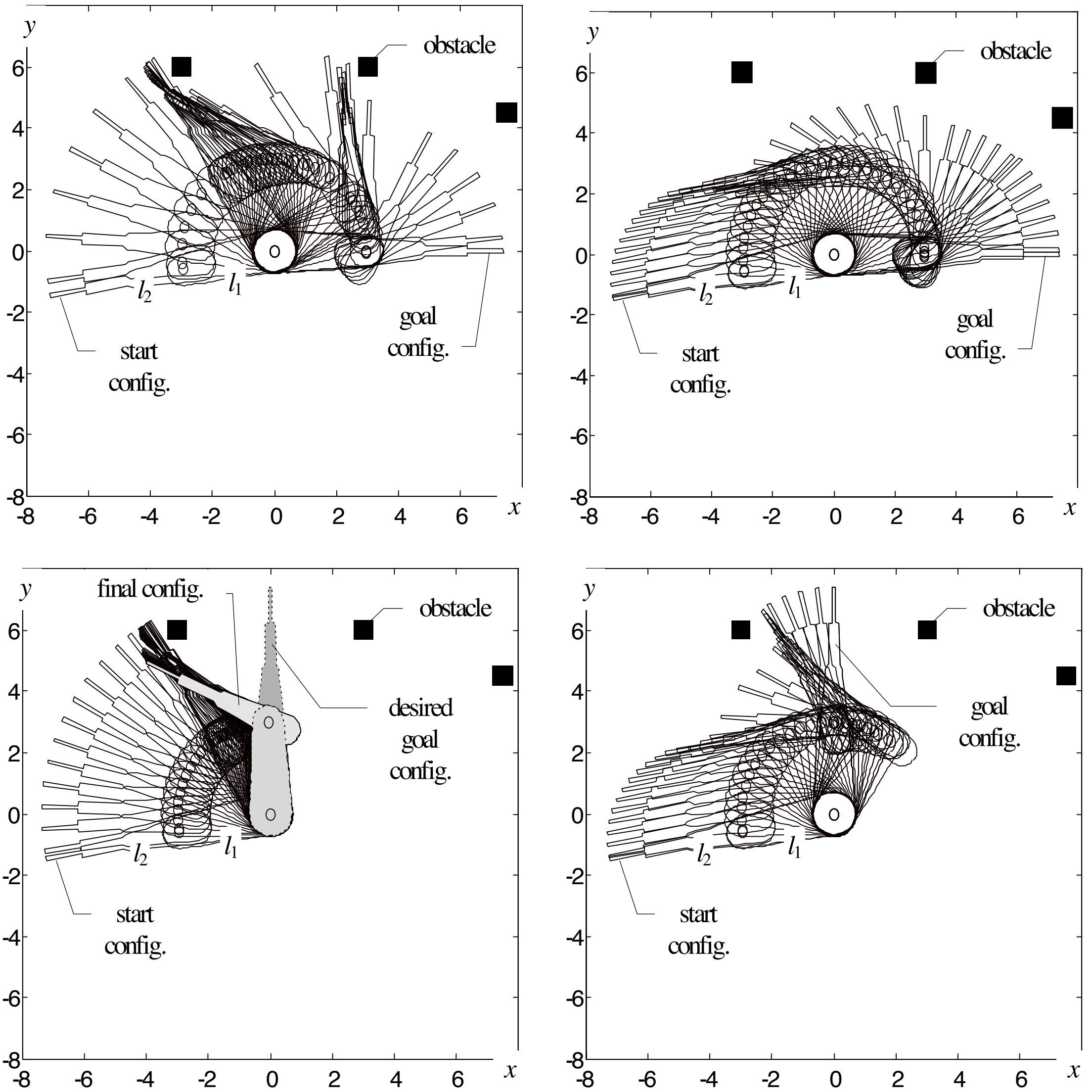


Figure 4.5-2: (top left) The manipulator is steered by the fuzzy navigator (total planning time: 6.3 seconds); (top right) The manipulator path is determined using the To&Fro algorithm described in Chapter 3 (grid size: 150×150, total planning time: 59 seconds); (bottom left) The manipulator is again steered by the fuzzy navigator. In this experiment, the fuzzy navigator gets trapped in a dead-lock situation and stops prematurely at the configuration denoted "final configuration"; (bottom right) In contrast to the navigator approach, the grid allows the computation of a path from start to goal (grid size: 150×150, total planning time: 60 seconds).

Although in these experiments the algorithm has not been applied directly to the real MA 2000 manipulator, the found path can be recorded and then applied to the real manipulator. Thus, the navigator here can be used as an off-line planning device. This fact also allows a comparison to the resistive-grid-based planning method presented in Chapter 3 (see Figure 4.5-2). Using the fuzzy navigator, the time to move the manipulator from start to goal in the experiment depicted in Figure 4.5-2 (top left) is 6.3 seconds (The same rule bases as in the experiment of Figure 4.5-1 were used.)

In Chapter 3, it has been found that the resistive grid needs not more than seven updates per node to construct a path in a very cluttered environment. Since the chosen example is of simple nature (for example link l_1 needs not to back up (Figure 4.5-2 (top right)) or only slightly (Figure 4.5-2 (bottom right)) three updates are sufficient to solve the two shown problems. Depending on the grid size, the time spent on updating an entire grid three times is 13 seconds (for a 75×75-grid) and 54 seconds (for a 150×150-grid (Figure 4.5-2 (top right and bottom right)), respectively. However, the planning time employing the resistive grid approach depends additionally on the workspace to C-space transformation (Chapter 2), placing the C-space obstacles in the grid (Chapter 2) and the gradient ascent in the updated grid (Chapter 3). Thus, the total time to plan this path increases to about 17 seconds in a 75×75-grid and about 60 seconds in a 150×150-grid, respectively. From this follows, that in this two-dimensional example the planning time using the fuzzy navigator or the resistive grid approach is of the same order. Note, however, that the computation time in the grid increases exponentially with the dimension (Chapter 3). This is not the case, if the fuzzy navigator is used (Sections 4.3.2 and 4.4.2).

In contrast to the resistive grid approach, the fuzzy navigator may get stuck in certain situations (see Figure 4.5-2 (bottom left) and (bottom right). However, the figure clearly shows that the navigator does not attempt to move the arm over the obstacle between link l_2 and target. This means that the system generates a collision-free path from start to goal in a large number of circumstances. However, even if it is trapped in a dead-lock, it behaves well and does not cause collisions with obstacles. If no valid path exists, the system behaves as if it were stuck in a dead-lock.

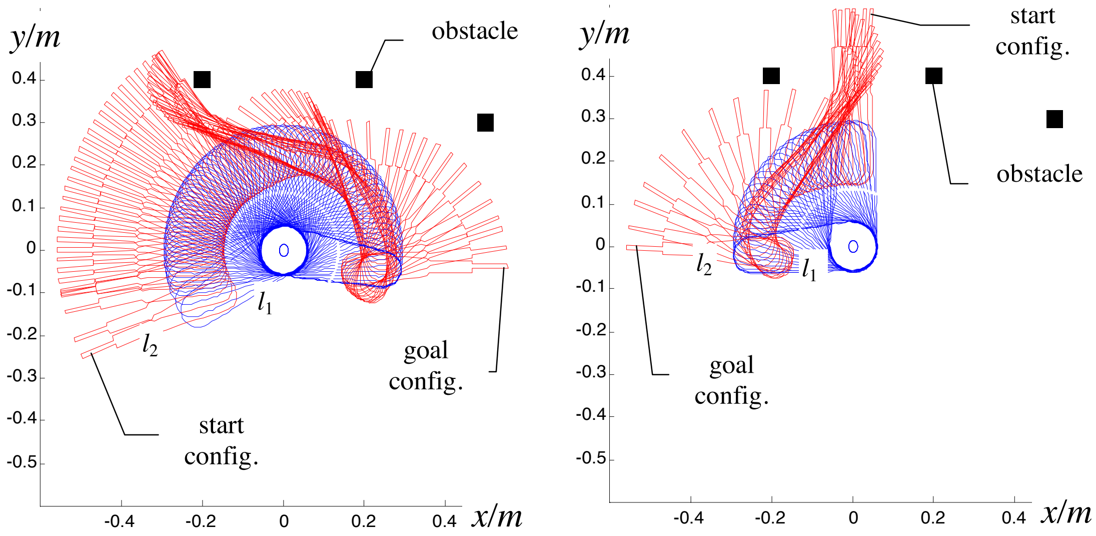


Figure 4.5-3: Depiction of the motion of the real manipulator.

4.5.2 Real-World Results

Initial real-time experiments have shown that the fuzzy navigator can be also successfully applied to a physical manipulator, the MA 2000 manipulator. The fuzzy navigator was implemented on the Transputer network (see Chapter 1 and [Althoefer94b]) that communicates with the MA 2000 [Zavlangas96]. In these real-time experiments, the navigator has been implemented in its basic form to investigate its properties in a real-world setting and to show its feasibility. In order to quickly acquire results, not all of the features described in previous sections have been implemented yet.

Firstly, the fuzzy-based technique has been applied only to the second link (elbow link), while the first link (shoulder link) moves with constant increments of its joint angle towards the goal configuration, assuming there are no obstacles in its range. The fuzzy unit of the elbow link receives at its two inputs the difference between actual and goal configuration as well as the signed distance between link and nearest obstacle. Secondly, the superimposition of the influence from the nearest obstacle on the right onto the nearest one on the left has not been considered here. The parameters for the fuzzy unit of the elbow link have been set up using the rule base constructed during the simulation experiments as a starting point and have been improved by trial and error.

The fuzzy navigation for the MA 2000 has been tested in a variety of environments including environments with moving obstacles. The results of two experiments are depicted in Figure 4.5-3. The model of the MA 2000 has been used to depict the motion of the real arm. That is, the sequence of angular values of the real arm's motion has been stored during the real-time experiment and was then applied to the robot's model. Further results are described in [Zavlangas96].

4.6 Reinforcement Learning

Even though the previous sections have shown that the fuzzy controller is capable of avoiding obstacles in both simulated (see Section 4.4) and real (see Section 4.5.2) environments, there are several reasons why the system needs to be made adaptive.

- The iterative process of finding the appropriate rule base by trial and error is time consuming.
- The rule base is different for different robot-arms and there is no simple relationship between the rules effective for one robot arm compared to another. (E.g. a long and narrow compared to a wide and short link.)
- The optimal fuzzy rule-base varies with the environment in which the arm is set to operate; compare for instance an environment that is sparsely populated with obstacles to a densely populated one.
- The learning scheme should be a general mechanism by which soft constraints on the operation of the arm can be introduced to allow continuous learning.

The aim of this section is to add an adaptation mechanism that meets these aims to the fuzzy navigator. In order to achieve any real gain in comparison to the trial and error development of the navigator, this adaptation process should perform a minimal amount of extra computations. Such considerations as well as the lack of advance knowledge of the obstacle locations rule out a supervised learning rule. Hence, a on-line reinforcement learning rule will be utilised (see [Mahadevan92, Sutton91, Gullapalli94, Millan92, Keerthi95]).

The adaptive algorithm is based on an *actor-critic* model. These models divide an adaptive controller in two parts. The first is the *actor* which performs actions in response to input from the environment. Depending on the changes in the environment after an action, the *critic* will send a reward (or punishment) signal to the actor to change its rule base such that the action will be more (or less) likely in the future. Here, the actor is the fuzzy controller and the critic is the new mechanism which adapts the fuzzy rule base. The critic receives information from the sensors and can be loaded with different rules that determine whether a sensory input represents a desirable situation or not.

The actor in the adaptive controller for a single link is identical to the controllers for one link in previous sections. It determines its actions on the basis of the distance to the nearest obstacle d_j and the target $\Delta\theta_j$. The critic can determine whether to punish or reward an action by inspecting the manipulator's performance. At each training step only the active rules are adapted with a reinforcement rule, while those rules which did not contribute to the particular movement remain unchanged. Three performance criteria are used for training: (1) distance between the link and obstacles, (2) the speed of the link and (3) the unlimited rotation of link l_2 around the end of link l_1:

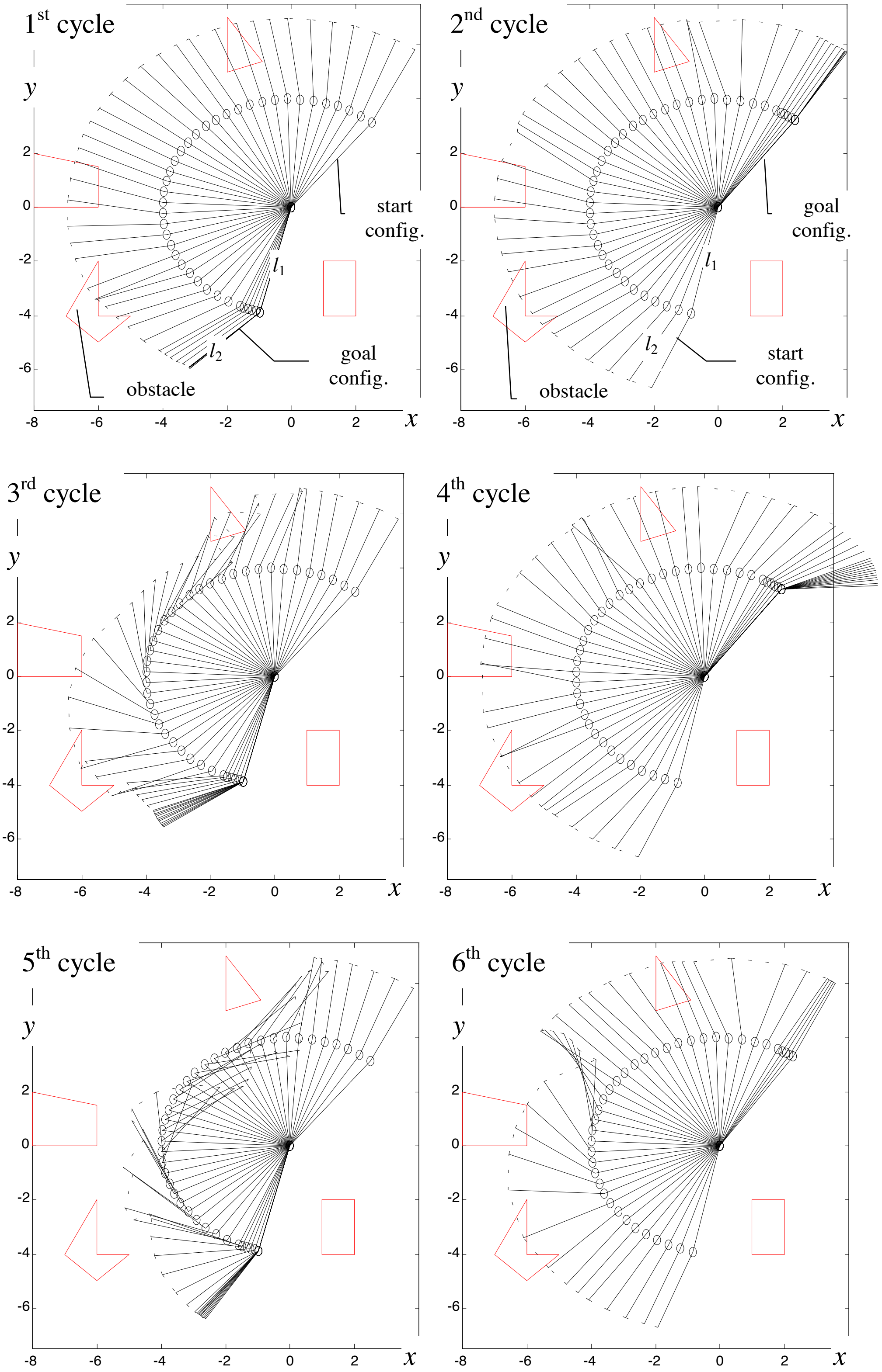
(1) Whenever the distance between link and obstacle falls below a given threshold, the action values of those active (= responsible) rules which proposed a movement towards the obstacle are decreased by a small amount (punishment), while those which propose a movement away from the obstacle are increased (reward). Once the navigator is trained and keeps the manipulator in sufficient distance to obstacles, the learning rule becomes inactive.

(2) The training with respect to the manipulator's speed is split into two parts: the controller is trained to avoid both very large and very small changes in the position of the manipulator. The first rule is useful to avoid big sudden jerks of the manipulator link. The second is active when the link virtually does not move at all. Again, active rules which support the desired performance are increased, those which are opposing it are decreased. Both rules are inactive when the manipulator performs correctly. A moderate speed is the desired aim of this training part.

(3) An unlimited rotation of link l_2 around link l_1 is not desired, since the motion of the links of the MA 2000 is limited to a range of approximately $\pm 0.75\cdot\pi$ and the learning experiments are expected to be transferred to the real-system. Thus, whenever the link l_2 tends to "collide" with link l_1 the rule base is appropriately altered. Again, an action which favours a motion in the expected direction (towards a stretched configuration of link l_2) results in the increase of the value, while an action which suggests the opposite motion (towards a collision with link l_1) is decreased. This rule is active only if the angle of link l_2 is greater than $0.64\cdot\pi$ or smaller than $-0.64\cdot\pi$.

The training here applies to the actions values in the rule base only. The fuzzification in the fuzzy navigator is carried out employing common sense and is not the target of the training method used here. For the following investigations a two-link stick-like arm as described in Section 4.4.1 was used. As in the previous section, only one link, viz. l_2, was controlled by a fuzzy unit, while link l_1 moved with constant steps to its goal configuration.

The above training strategies were used in the experiments which are presented in Figure 4.6-1. The influence or strength of the individual strategies were adjusted during many training runs until a satisfactory performance and a stable rule base developed. While conducting the experiments it became obvious that all three training criteria influenced the rule base of the controller. Moreover, the combined influence of all three criteria provided the desired manipulator performance, while changing the strength of one of them could lead to undesired values in the rule base. Training to achieve a goal directed behaviour was not investigated in these initial experiments, still the manipulator reached in most cases the goal configuration.



(see caption next page)

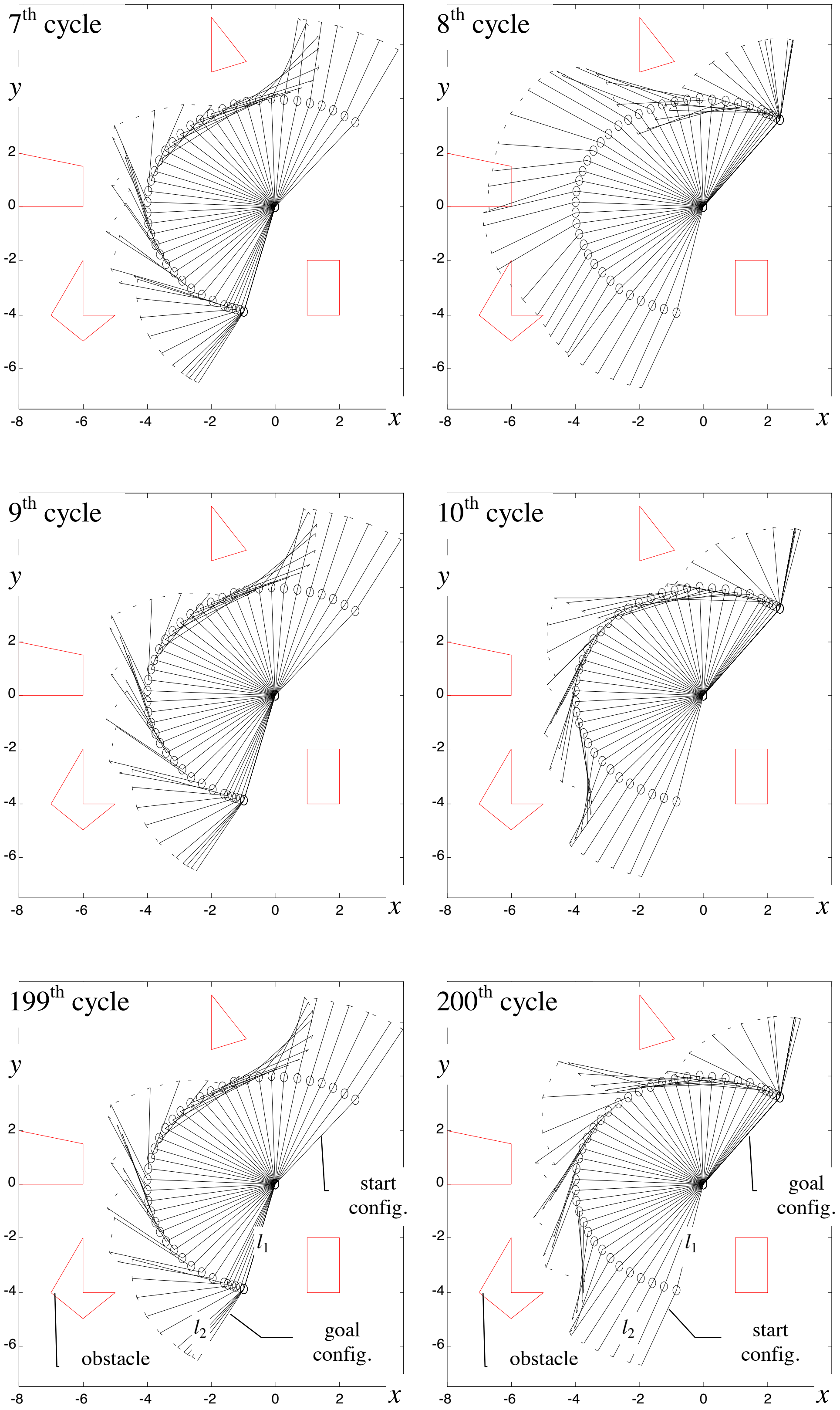


Figure 4.6-1: Training of the fuzzy controller in the fuzzy unit of link l_2. The training is shown at different training cycles. After about ten cycles the initially untrained rule base stabilised. Note that in every second training cycle start and goal configuration were swapped to train the rule base for obstacles and target configurations on both sides.

Training occurred every time the arm moved. As shown in the following figures, at every instance of the manipulator motion the whole training procedure was applied. This means that the fuzzy navigator was trained approximately 50 times between start and goal. If after 50 iterations the goal was not reached, the training was interrupted and a new training cycle was started. Every second training cycle start and goal configuration were swapped. This was done to train the fuzzy controller in both directions.

Figure 4.6-1 shows that already after ten complete training cycles a convincing performance was achieved, although each rule of the initial rule base was set to a small constant value of 0.001. After these ten cycles were complete, training continued but the performance of the manipulator changed only marginally and the actions of the rule base experienced only minor changes. That is, the rule base stabilised.

A second experiment was conducted where the same training strategies were used, but the initial rule base was built using common sense. This experiment was meant to check whether the training provides improvement and at the same time keeps the overall structure of the original rule base.

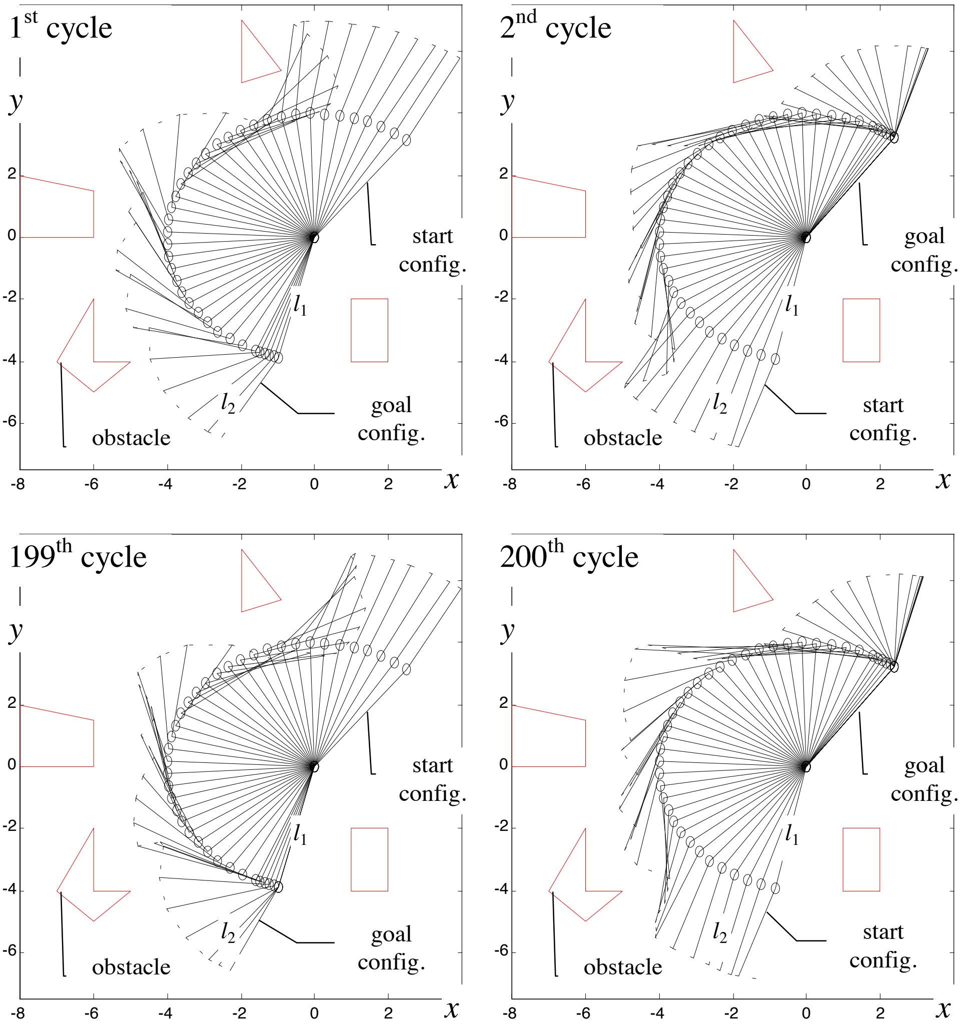


Figure 4.6-2: Training of the fuzzy controller in the fuzzy unit of link l_2. Here the rule base was initially constructed using common sense. The training is shown at different training cycles. Only small changes in the manipulator motion from the beginning of the training to its end can be seen. Again, in every second training cycle start and goal configuration were swapped.

Figure 4.6-2 shows the manipulator motion at the beginning and end of the training. The base stabilised after only a few training cycles and did not change very much while the training continued. In general, the manipulator motion has changed only slightly from the first to the last training cycle. When steered by the original controller, link l_2 of the manipulator came occasionally close to obstacles (see Figure 4.6-2 (top)). This aspect has been dealt with successfully as shown in Figure 4.6.2. (bottom). Whether the other two strategies were active is difficult to say by looking at the manipulator motion. It is possible that the original motion did not fall in the range of these two strategies. The comparison of the two rule bases (initial and after training) clearly shows that the influence of the learning process was only in places (see Table 4.6-1 and Table 4.6-2). Although this training brings about a continuous change of the action values, they are still interpretable. For example, "-0.6238" represents an actuator command which is "*bigger right*" than "-0.4188".

μ_2 \ ν_2	*far left*	*left*	*clse left*	*clse right*	*right*	*far right*
far left	0.1000	-0.3000	-0.5000	0.3000	0.3000	0.3000
left	0.1000	-0.0010	-0.5000	0.3000	0.3000	0.1000
contact	0.0000	0.0000	-0.1000	0.1000	0.0000	0.0000
right	-0.1000	-0.3000	-0.3000	0.5000	0.0010	-0.1000
far right	-0.3000	-0.3000	-0.3000	0.5000	0.3000	-0.1000

Table 4.6-1: Initial rule base based on common sense rules.

μ_2 \ ν_2	*far left*	*left*	*clse left*	*clse right*	*right*	*far right*
far left	0.1182	-0.4188	-0.6238	0.3000	0.3000	0.3000
left	0.0644	-0.4099	-0.6365	0.3001	0.3117	0.1096
contact	-0.0258	-0.4052	-0.2634	0.1619	0.3904	0.1265
right	-0.1000	-0.3032	-0.3000	0.6093	0.3546	-0.0562
far right	-0.3000	-0.3000	-0.3000	0.5018	0.2391	-0.1787

Table 4.6-2: Rule base after 200 training cycles.

It seems that the learning method is a valuable tool to adapt the actions of a common-sense rule base or a pre-trained rule base to a particular environment.

One very important problem has been neglected so far: how will the navigator perform if confronted with a new environment. As one could see in earlier sections, the rule base using common sense action values was able to cope with different and even dynamic environments. Initial experiments in this direction indicate that the fuzzy controller trained from scratch is able to cope with slight changes in the environment, but encounters problems when confronted with completely new environments. Here, obstacle avoidance is no problem, while reaching the goal configuration is. This might be due to the fact that goal-reaching was not part of the training agenda.

In a further experiment, the adaptive navigator was applied to the MA 2000 manipulator. To avoid damage of equipment, the learning took place on a simulated arm in an environment modelled after the real environment. The training approach here has been expanded by an extra strategy which punishes the rule base for not reaching the goal. After 50 training cycles, the simulated arm achieved a reasonable performance and the controller was transferred to the real arm. Although the simulated manipulator and the environment were a good model of the real world, and the simulated manipulator moved without collision from start to goal after training was completed, problems occurred when the trained rule base was applied to the physical device. Again, obstacle avoidance was carried out successfully by the navigator, but near the goal configuration link l_2 started to oscillate and performed several up-and-down movements before stopping at the desired goal configuration.

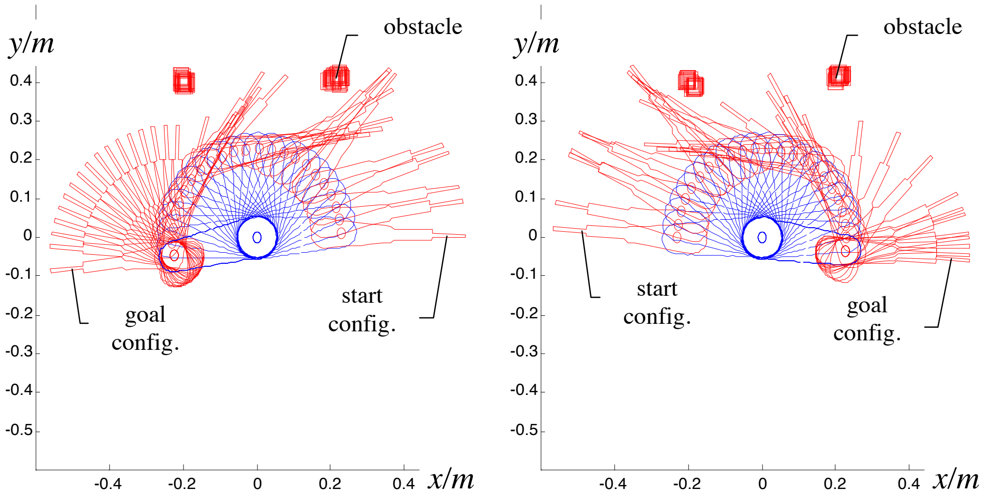


Figure 4.6-3: A typical training sequence where the fuzzy navigator “learns” to steer the MA 2000 collision-free through its workspace. Randomly moving obstacles as well as randomly changing start and goal configurations are used to help the controller adapt to different environments (generalisation). These training snapshots were taken after 29 (left) and 30 (right) training cycles, respectively. The right figure shows that the arm can start oscillating near the goal configuration.

These first results are very promising, when one takes into consideration the fact that the action values in the rule base were generated entirely by a learning algorithm. Future work should aim on the incorporation of the training into the real-world system. This would allow the navigator to adapt the manipulator’s motion to changes in the environment in real-time.

4.7 Summary and Discussion

This chapter presented a novel fuzzy-based algorithm able to generate collision-free paths for robotic manipulators. The proposed control system is divided into separate fuzzy-units which provide actuator commands to the individual links of the manipulator. Each fuzzy-unit receives at its inputs signals which represent firstly the distance between link and nearby obstacles and secondly the difference between actual configuration and target configuration. The fuzzy-rule-bases were generated using common sense rules which were then refined by trial and error as well as by means of a learning algorithm. Two main considerations were taken into account in the construction of the rule base of such a fuzzy unit: on the one hand, the rule base has to exert a strong repelling influence on the link when it approaches an obstacle and, on the other hand, to impose an attracting influence which emanates from the target configuration. The rule base has to evaluate the interaction of the often contradictory inputs and produce an appropriate motor command. Thus, it combines a goal-directed behaviour with an obstacle-avoidance behaviour.

In contrast to mobile robots which can be considered as single moving objects, manipulators consist of multiple moving objects which influence each other. In the case of a non-branching manipulator with a fixed base, the motion of proximal links obviously influences the more distal ones. In this application, fuzzy-units of proximal links are not only informed about obstacles in their vicinity, but also about obstacles in the vicinity of more distal links. Thus, an obstacle near to a distal link also provides a repelling influence to more proximal links. This reduces the likelihood that distal links whose motion strongly depends on the motion of proximal links collide with their nearby obstacles.

The presented fuzzy navigator may be improved by increasing the information transfer between distal and proximal links. In addition to the information passed from units of distal links down to units of proximal links informing about the distance between links and obstacles, the units of distal links could also inform the units of proximal links about their inability of reaching the goal configuration (unreachable situation or dead-lock). Thus, in a dead-lock situation, proximal links may continue their motion even though they have reached their goal configuration, in order to help distal links to leave the dead-lock area and reach their goals.

In the experiments carried out, the fuzzy controller has been applied to different manipulators in a variety of environments. It was shown that the fuzzy controller can cope with dynamic environments. Furthermore, the rule bases which had been developed for a particular manipulator type needed only minor changes to be able to successfully steer another manipulator with different proportions.

The proposed fuzzy system can be considered as a reactive system. Once the rule base is constructed, the system is equipped with a set of instructions which enable it to deal with many obstacle constellations. Although the fuzzy system is able to construct paths from a start to a goal configuration, planning as discussed in the previous chapter is not carried out. Planning a path presupposes global spatial knowledge which the fuzzy navigator does not have. It only can plan one step ahead. At each iteration, it makes a guess as to which is the best actuating command to be sent to the manipulator so that the manipulator’s state is altered in such a manner that it comes closer to the goal state. Any state change experienced by the manipulator is interpreted by the fuzzy navigator as a new environmental situation to which the navigator reacts with a new actuating command. The experiments conducted show that the fuzzy navigator provides collision-free paths in many cases, even though the navigator is a simple reactive mechanism with only a small rule base.

Despite its local nature the fuzzy-based system can outperform a global planning strategy in many applications. The main advantage over the global planner is that the fuzzy system can be implemented on-line. This allows the fuzzy system to adapt to changes in the environment caused for example by moving obstacles. Since the environment is newly conceived at each iteration, any change occurring in this environment is easily incorporated into the next computation. Global planning strategies are usually hampered by lengthy computation times and therefore can not adapt to fast changes occurring in the robot’s workspace.

Moreover, the fuzzy system in its presented form does not suffer from the curse of dimensionality observed in most planning strategies. Thus, it can be applied to manipulators with a high degree of freedom. The main drawback of the suggested strategy is that it can get trapped in particular obstacle constellations (dead-lock). Solutions for this problem are discussed in Chapter 5.

The rule-based fuzzy approach convinces with its transparency and allows the designer to shape the decision process by altering rules individually so as to achieve a specific performance. There has been an increased interest in combining fuzzy systems with neural networks because fuzzy neural systems merge the advantages of both paradigms. On the one hand, parameters in fuzzy systems have clear physical meanings and rule-based and linguistic information can be incorporated into adaptive fuzzy systems in a systematic way. On the other hand, there exist powerful algorithms for training various neural network models. The successful introduction of learning techniques to fuzzy-based algorithms which control mobile robots has been reported in literature [Kosko92, Tschichold96, Hoffmann96]. One of these techniques, reinforcement learning, has been applied to the fuzzy-system described here.

The learning method is used to either generate a rule base from scratch or refine a coarsely set-up rule base. In the training phase, the manipulator or - to avoid damage - its simulated counterpart is steered by the fuzzy-algorithm through different environments. If the initial rule base has been set up by employing common sense rules, training times are shorter than in cases neural networks are employed where training begins always at a random state. During training the fuzzy-controller adapts to the given environment and its rule base is still interpretable.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The thesis has proposed and verified the suitability of novel motion planning strategies for robotic manipulators.

Building on and complementing each other, the neural-network-based workspace to C-space transformation and the fast path planning strategy based on the neural resistive grid have been shown to produce good manipulator trajectories. This approach to motion planning always finds a path when one exists. It is hampered by the dimensionality problem of C-spaces which increase exponentially with the degree of freedom and lengthy computations prior to the robot movement, the latter making them able to perform satisfactorily only in static or slowly changing environments. The fuzzy navigator has been introduced as a technique capable of tackling these issues. Although this local navigation technique can get caught in so-called dead-locks caused by particular obstacle constellations, it has many advantages: it adapts to fast changes in the environment, does not suffer from the curse of dimensionality, and outperforms a global planning strategy in many applications its main advantage being its on-line implementation.

The following sections discuss the main points of the research work presented.

5.1.1 Workspace to C-space Transformation for Robotic Manipulators

The neural-network-based transformation technique presented in this thesis has been shown to be a suitable tool to construct the configuration space of robotic manipulators. The technique has been successfully applied to manipulators with revolute and prismatic joints. The basic transformation technique developed for two-link arms has been expanded to compute the C-space patterns for planar manipulators with *n* revolute joints. Experiments conducted using the MA 2000 manipulator proved the technique's real-world feasibility.

The radial-basis-function neural network used for the transformation process has been shown to learn the highly non-linear mapping between obstacle points and C-space patterns. The interpolating capabilities of the network allow the construction of those C-space patterns, which occur in response to unknown input stimuli, with only a small error. The memory requirements to store the weights of the network were particularly small compared to those reported in [Newman91, Branicky90]. Although the focus in Chapter 2 was on planar manipulators, the technique can be applied to three-dimensional manipulators as shown in Chapter 3.

5.1.2 A Neural Resistive Grid for Path Planning

The use of a computer-emulated neuro-resistive grid to carry out manipulator path planning in C-space has been proposed and investigated. The To&Fro algorithm introduced in Chapter 3 has been proved to rapidly generate an activity distribution in the grid suitable for path planning. This activity distribution has its unique maximum at the node representing the goal configuration and following the greatest gradient from a start node results in a collision-free path ending at the goal configuration.

The update sequence of this algorithm is especially adapted to the shape of the C-space obstacles usually occurring in a manipulator’s C-space and spreads the activity in the obstacle-free regions of the C-space approximately eight times faster than conventional update sequences described in literature [Cichocki94, Glasius94]. Early termination is another important feature of the To&Fro algorithm. The algorithm terminates once the start node or one of its neighbours experiences a rise in activity - long before the nodes’ activities converge. For example, the number of updates per node in a two-dimensional grid necessary is a constant value: five to seven. The number of total updates necessary increases linearly with the number of grid nodes. Comparing the latter aspect with the fact that the number of updates necessary to yield a converged solution increases with the square of the number of nodes, clearly shows the suitability of the To&Fro algorithm for path planning.

The path found in an activity distribution due to the early interrupting To&Fro algorithm is free of local extrema, and therefore, a collision-free path is found if such path exists. The algorithm is resolution-complete, that is, it always provides a collision-free path from a start state to a goal state, if such path exists.

Two different boundary conditions, the Dirichlet and the Neumann boundary condition, have been investigated. It has been shown that a computer-emulated grid due to the Dirichlet boundary condition provides more quickly a solution suitable for path planning than a grid which is updated employing the Neumann boundary condition.

The To&Fro algorithm has been also compared to the A*-algorithm [Latombe91]. The To&Fro algorithm has been shown to be faster than the A*-algorithm in many situations. Although the To&Fro algorithm does occasionally generate a path which is not the shortest one, the path is always collision-free. In many path planning problems finding a path in short time is preferable to the more time-expensive search for the shortest path. Since the centre of investigation of this chapter was on finding collision-free paths in short time, the To&Fro algorithm can be said to provide better results than the A*-algorithm.

The grid-based strategy has been extended to carry out path planning in non-homogenous and non-topologically ordered graphs. This extension can also be used to add soft safety margins around obstacles. This safety margin assures that the robot keeps in most situations at least a minimum distance to the obstacles; only where the safety margins of obstacles merge it is soft, i.e. penetrable, allowing the robot to move through narrow gaps.

The application of the neuro-resistive grid to a physical manipulator, the MA 2000, has been successfully carried out. The experimental studies clearly showed the feasibility of the suggested method in real-world applications.

If a discrete representation of a state space, as for example a discretised C-space, a topologically ordered map or any kind of graph, is available, the resistive grid using the described To&Fro algorithm is probably one of the fastest of the global planning methods to calculate a path. This is the case for resistive grids which are run on computers, and especially for those implemented in specialised hardware. Thus, the neuro-resistive grid presents a strong alternative to other methods which are still in use in the robotics research community (e.g. the A*-algorithm or potential field methods combined with random search mechanisms).

5.1.3 Fuzzy-based Navigation and Obstacle Avoidance for Robotic Manipulators

A novel fuzzy-based technique able to navigate a robotic manipulator collision-free in static as well as dynamic environments has been presented. The proposed technique can be seen as a composition of separate mechanisms, called fuzzy-units, which steer the manipulator links individually. Each fuzzy-unit comprises a fuzzy-controller whose rule base is set up to exert impelling forces which depend on the local environment of the link. Two main components of these forces can be distinguished. The first force component pushes the link towards the target configuration while the second component makes the link dodge obstacles. Thus, the rule base combines a goal-directed behaviour with an obstacle-avoidance behaviour by appropriately evaluating the interaction of the two aforementioned aspects of navigation.

Early experiments showed that the control of the manipulator links in a completely independent fashion was not suitable. It has been found that the fuzzy units which control proximal links have to be informed about obstacles in the vicinity of distal links. Thus, an obstacle near a distal link influences the fuzzy units of proximal links in such a way that the latter ones change their motion appropriately. This reduced the likelihood that distal links whose motion strongly depends on the motion of proximal links collide with their nearby obstacles.

Furthermore, it has been found that the fuzzy units should not base their decision on *a single* nearest obstacle only, but on the *two* nearest obstacles which are on the left and right of a link. To achieve this, the fuzzy units calculated separately a solution for both obstacles. These outputs were then superimposed and applied to the actuator of the corresponding link. This approach was particularly useful in situations where manipulator links were situated in a narrow gap between obstacles. When using the approach without superimposition the links in the gap tended to oscillate, while the improved approach provided smooth manipulator movements.

The fuzzy-based technique has been successfully applied to different manipulators operating in real as well as simulated environments. The technique has been proved to work without complications in a variety of dynamic as well as static environments. It has been also shown that the rule bases set up for a particular manipulator type needed only minor changes to be able to successfully manoeuvre a manipulator with different proportions. Occasionally, the manipulator did not reach the desired goal configuration due to a dead-lock situtation, however, in most experiments carried out the navigator steered the manipulator successfully around obstacles.

In most of the experiments shown the rule bases of the fuzzy units were generated using common sense and then improved by trial and error. The rule-based approach convinced with its transparency. A specific performance for each fuzzy unit was easily produced by altering rules individually. In later experiments an on-line reinforcement learning mechanism had been introduced to automatically adapt the fuzzy rules to certain constraints, such as keeping a minimum distance to obstacles, limited angular step width, avoiding collision between links, etc. All experiments involving learning indicated that the proposed learning mechanism converged to a stable solution. The learning mechanism has been used to generate a rule base from scratch as well as to refine a rule base which had been coarsely set-up by common sense. The latter approach was particular promising, because only a short training time was necessary to achieve an improved rule base. Since the learning mechanism works on-line a continuous adaptation to changes in the environment is possible. At all times the commands of the rule bases were interpretable.

5.2 Future work

The results produced by the investigation in this thesis of the resistive-grid-based method and the fuzzy-based navigator provide a solid basis for further research and can be used as a spring-board for moving to new directions.

5.2.1 Hybrid System

The next step in the search for more efficient path planning strategies could be the combination of the motion planners described in this thesis with aim to create a versatile planning system. The following approaches can be envisaged.

One possibility is to design a planning system with modules on different hierarchical levels [Althoefer94b]. One could imagine a low-level module to be a fuzzy-based navigator (Chapter 4) which receives signals from a global planning module on a higher level in the hierarchy. The planning module could be based on the workspace-to-C-space-transformation technique of Chapter 2 and the neuro-resistive grid strategy of Chapter 3. Before the manipulator motion takes place, the planning module computes a manipulator trajectory which can be seen as a sequence of via-points from a start to a goal configuration. These via-points are sequentially fed to the navigation module which provides a secure path from one via-point to the next. Since the latter module works on-line, changes in the environment can be dealt with.

Alternatively, a path planning approach could be imagined where the fuzzy navigator and neuro-resistive grid collaborate - each compensating for the others handicap. Initially, the navigator attempts to move the simulated counterpart of the manipulator from a start to a goal configuration. Whenever a dead-lock or unreachable situation is encountered, the simulated manipulator performs random movements in the vicinity of the dead-lock and a C-space map is constructed in which configurations which are safe and those which result in collisions are registered. Then, the neuro-resistive grid can be invoked to plan a path in this map to leave the area of the dead-lock. From there the navigator can take over control again. The two techniques can alternate until the goal configuration in the simulator has been reached and the found path can be applied to the physical manipulator. On the one hand, this approach offers a solution to the dead-lock problem of the fuzzy navigator. On the other hand, the C-space maps constructed during the operation of this approach are confined to a small subspace of the complete C-space and, thus, the dimensionality problem is avoided.

Another hierarchical approach could be envisaged which makes use of two fuzzy-based systems on different hierarchical levels. The fuzzy system on the more local level could be based on the fuzzy navigator described in Chapter 4. The fuzzy system on the higher level could be a planning mechanism whose rules reflect a more general and abstract knowledge about the obstacle distribution in the manipulator's workspace in order to deal with the dead-lock situation of the low-level navigator. A rule in such a fuzzy planner could be for example as follows: "If target is behind obstacle which is near second link and first link is already in goal configuration, move first link over the goal configuration". The rules of the high-level fuzzy planner could be developed during test runs in different environments. Whenever the low-level navigator gets stuck in a new dead-lock, a new rule has to be added to the rule base of the high level fuzzy planner. The process of adding new rules may be automated. It has to be tested, whether this approach is capable of dealing with any obstacle constellation. The approach would definitely represent an improvement over the original fuzzy navigator on its own.

Another way of improving the fuzzy navigator could be to increase the information transfer between distal and proximal links. In the described navigator system (Chapter 4), units of distal links provide units of proximal links with information about the distance between links and obstacles. In an advanced approach the units of distal links could also inform the units of proximal links about their inability of reaching the goal configuration (unreachable situation or dead-lock). If such a situation occurs, proximal links may continue their motion even though they have reached their goal configuration, so that distal links can leave the dead-lock area and reach their goals. This advanced approach will help to reduce the number of dead-locks.

5.2.2 Implementational Aspects

In this thesis, all algorithms introduced were implemented on single-processor machines. However, due to their structure an efficient implementation of the algorithms in parallel can be easily realised.

The algorithm described in Chapter 2 transforms pixels representing a workspace obstacles into a C-space representation. One could imagine a multi-processor machine where each obstacle pixel is processed in parallel by individual processing units. Thus, the transformation time would only depend on the time necessary to transform a single pixel and *not* on the number of obstacle points in the workspace. To achieve a fast mapping into a discrete C-space, memory should be shared by the processing units allowing parallel writing access to the memory. A further acceleration in terms of speed could be achieved by processing the C-space patterns of the subarms of a manipulator in parallel as outlined in Chapter 2. Also, the neural network used for the transformation could be implemented on a parallel processing computer.

The resistive grid approach presented in Chapter 3 is especially suited for an implementation in VLSI. Experiments and investigations in this area show that processing times in the order of micro seconds can be achieved [Roska96, Koch96]. The main problems to be solved in this area are the high interconnectivity in the grid to compute high dimensional configuration spaces as well as the connection to the computer periphery which writes to and reads from the hardware grid. The on-going research in the fields of VLSI implementations of neural networks [Roska96] and image processing methods [Koch96] may very well contribute to the area of path planning in robotics.

Also the fuzzy navigator proposed in Chapter 4 can be implemented on a parallel processing machine. Each fuzzy unit which controls an individual manipulator link could be processed by a separate processor. Each processor should do a two-step computation. In the first step, each of these fuzzy processors should calculate the distance between link and nearest obstacle(s), since fuzzy processors of proximal links incorporate the link-to-obstacle distance of distal links in their calculation and this is the only dependence between the fuzzy processors. In the second step, all processors can compute independently and in parallel the appropriate motor command. Moreover, each fuzzy unit might be broken down into subunits. For example the fuzzyfication of the different inputs might be carried out in parallel.

5.2.3 Sensors

The main focus in this thesis was on the development and investigation of planning and navigation techniques for robot manipulators. In order to create an autonomous robot system, it is essential that this is system is able to acquire information about the robot's environment. A variety of sensors have been developed over the years which are suitable for different manipulator tasks [Indyk94, Latombe91]. Most promising is the use of camera systems for manipulator path planning tasks. In this thesis, initial experiments have been carried out successfully employing a camera which acquires the two-dimensional workspace of a planar manipulator (Chapter 2). A next step could be to tackle the problem of moving a three-dimensional manipulator in a three-dimensional workspace. To solve this problem one could use a multi-camera system which oversees the manipulator's workspace [Ritter92]. This multi-camera system could be supported by one or more cameras mounted on the manipulator. Here, the main problems are the transformation of the multiple two-dimensional camera outputs into a three-dimensional representation and the correct identification of hidden obstacles or hidden obstacle parts. Promising techniques have been suggested to deal with the latter problem in environments where mainly known objects occur by using a library of objects to improve the identification of obstacles [Jaitly96, Jaitly96c].

To turn the fuzzy-based navigation system of Chapter 4 into a real-world application, ultra-sonic scanners could be mounted on the manipulator links as proposed in [Risse95]. This sensor system is especially suitable for this kind of problem, since only the information in the vicinity of the manipulator is needed and usually a detailed representation of the environment is not necessary.

5.2.4 Transformation of Complex Obstacle Primitives

The process of the workspace to C-space transformation, may be accelerated by transforming complex obstacle primitives such as lines, planes, circular obstacles and combinations of these primitives [Newman91, Branicky90]. To achieve this, known complex obstacles which are present at pre-known locations and orientations, can be grouped in libraries. Prior to the transformation, the obstacles in the workspace representation are matched against library models. The library models with descriptors similar to those of the workspace obstacles and a sufficient level of confidence for a match are chosen [Jaitly96]. The output of the matching process describes obstacles in terms of distance, size and orientation. A network trained to react to such features with the appropriate C-space pattern is capable of transforming a complex obstacle in a single computation. However, it has to be tested whether this more sophisticated approach will result in a reduction in transformation time when compared to the obstacle point transformation, since the computational load is now increased on the imaging side of the overall transformation process.

Appendix A

This appendix shows the manipulator (MA2000) which had been used throughout the thesis for the real-world experiments. Furthermore, it shows the recorded C-space patterns for shoulder and elbow link (finger fixed) and the C-space patterns for elbow link and finger (shoulder link fixed).

Appendix A-1

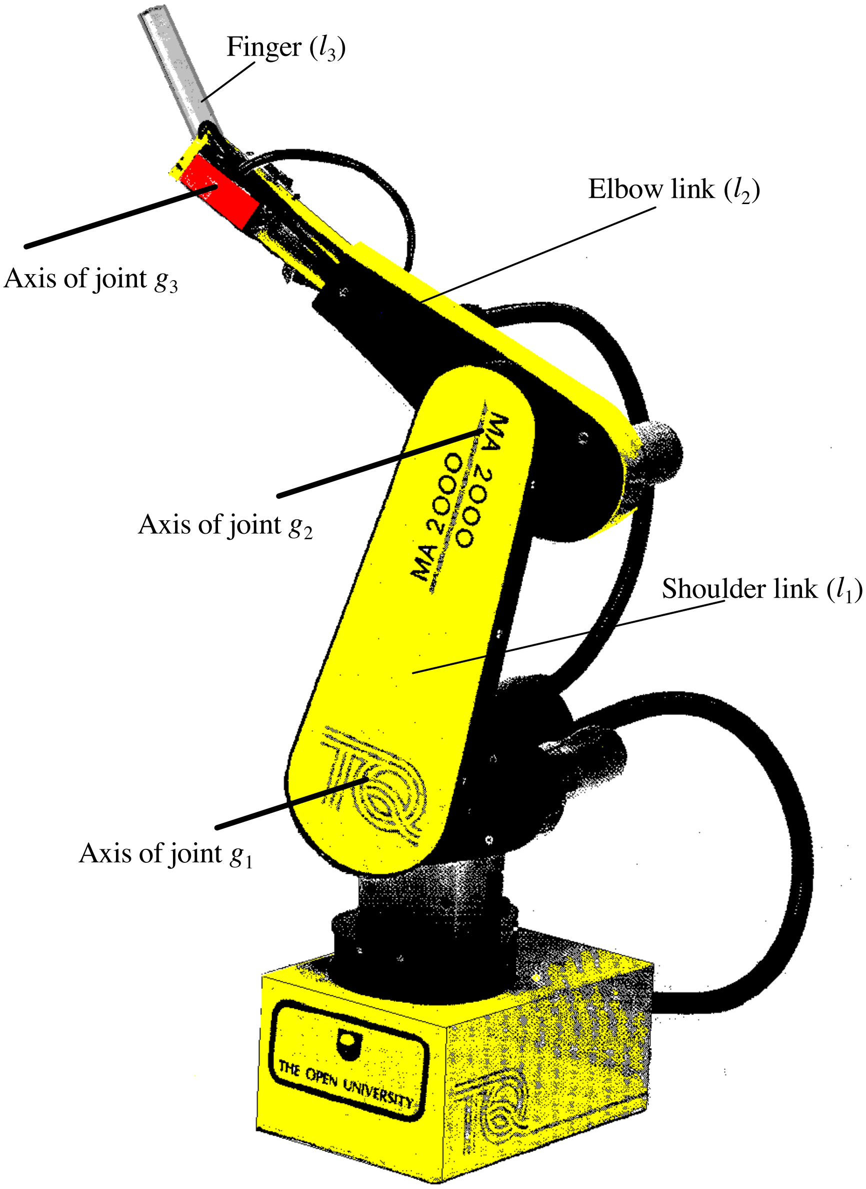


Fig. A-1.1: The manipulator MA2000. The active joints are denoted with g_1, g_2 and g_3. Their axes are parallel to each other.

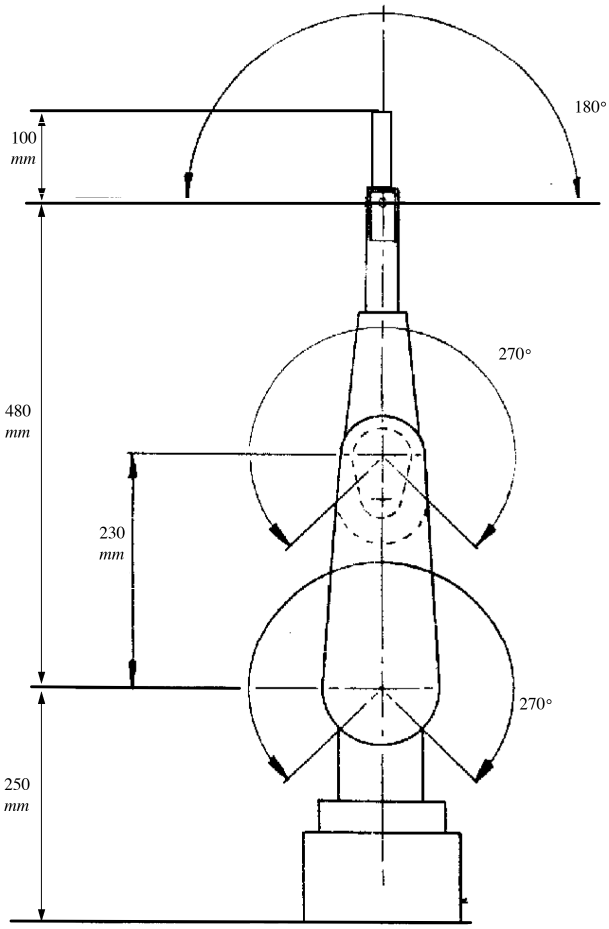


Fig. A-1.2: Fully extended position and maximum travel of the MA 2000 (after description by TecQuipment).

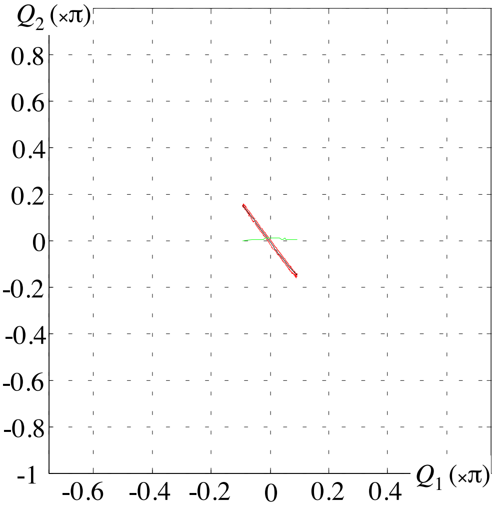
Appendix A-2

The following figures show C-space patterns due to collisions between an obstacle rod and the perimeter of the shoulder (l_1) and elbow link (l_2) of the MA2000-manipulator at increments of distance d. For all these measurements, the joint connecting the finger to the elbow link has been kept at 0°. The measurements have been carried out by moving the two active links of the MA2000-manipulator around the obstacle ensuring that at all times some point of the manipulator was in contact with the obstacle.

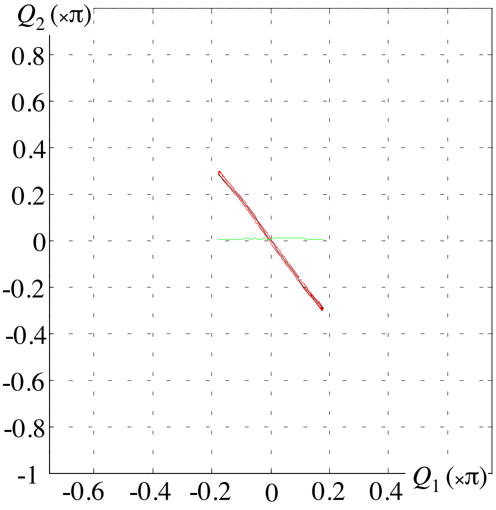
The obstacle rod has a diameter of 3 *mm*. The distance between the centre of the axis of joint g_1 and the centre of the rod is shown in the header of each figure. For obstacles in the range d=[278 *mm*, 548 *mm*], there are collisions with link l_2 only. For smaller values of d (d=[159 *mm*, 273 *mm*]) there are collisions with link l_1 and l_2.

Each figure shows the recorded C-space pattern, the centre line as well as the distance in vertical direction between the centre line and the recorded C-space pattern (dotted). The grey-shaded blocks describe areas which include collisions due to link l_1 and collisions due to the back of link l_2. In the path planning experiments described in this thesis, those areas are set to forbidden regions.

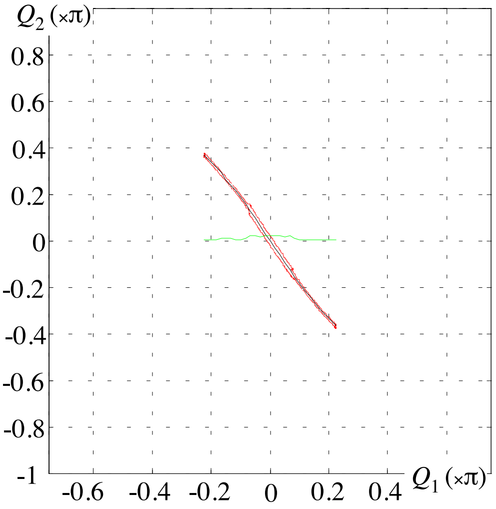
C-obstacle (distance: 548 mm)



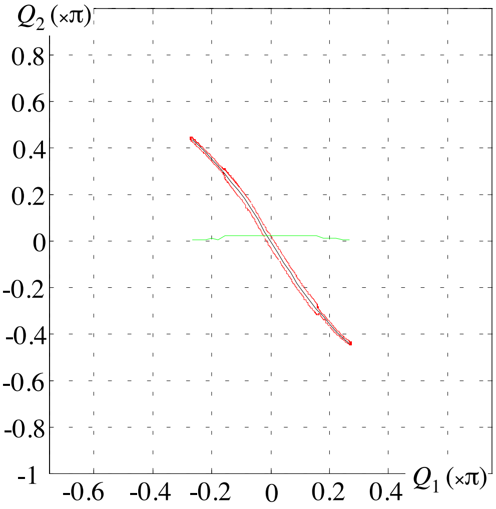
C-obstacle (distance: 508 mm)



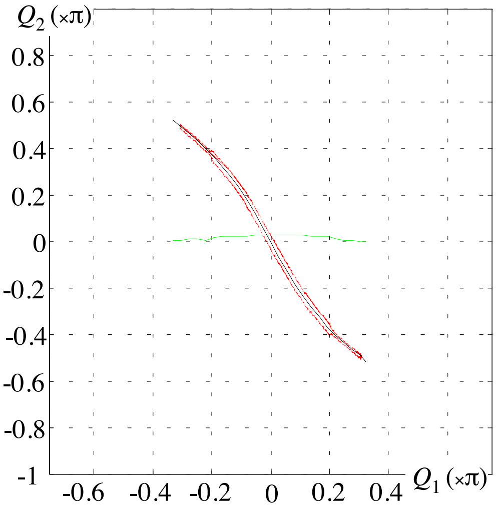
C-obstacle (distance: 468 mm)



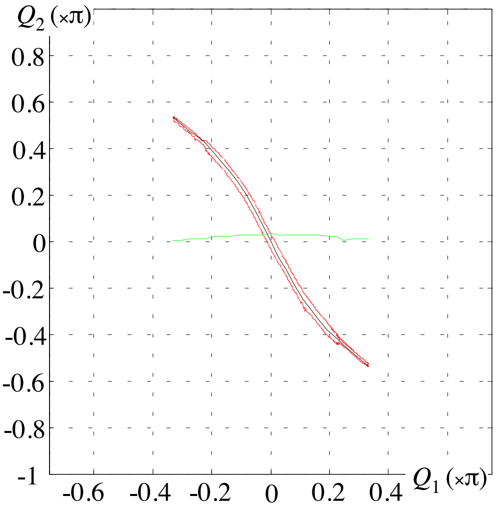
C-obstacle (distance: 438 mm)



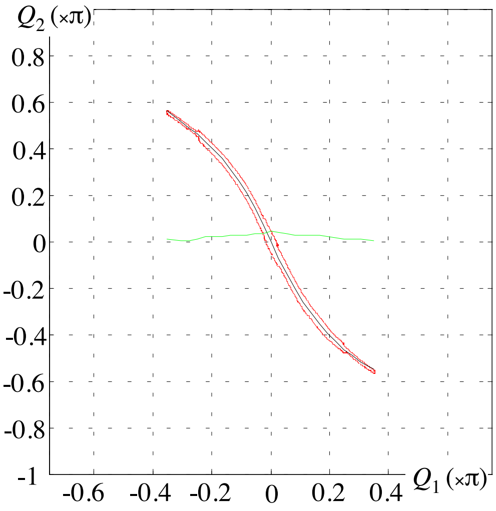
C-obstacle (distance: 408 mm)



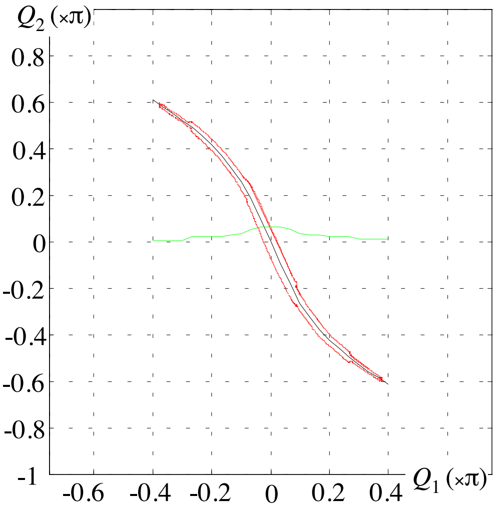
C-obstacle (distance: 388 mm)



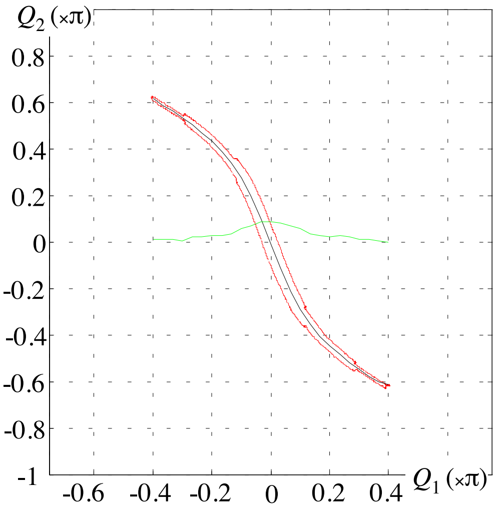
C-obstacle (distance: 368 mm)



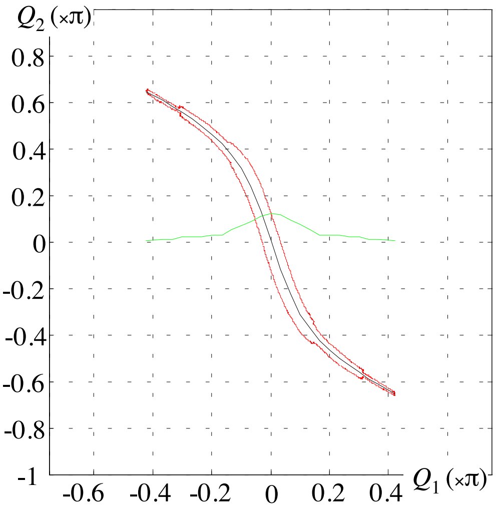
C-obstacle (distance: 348 mm)



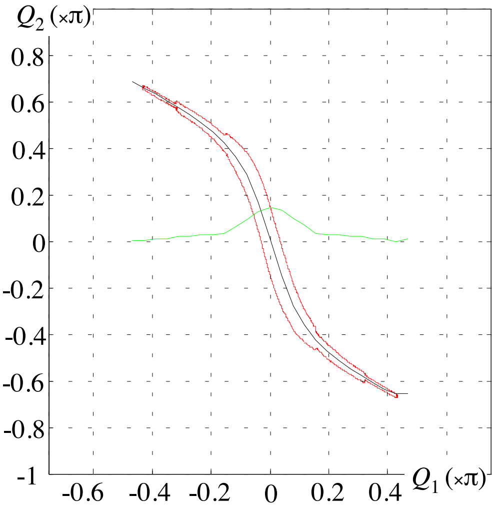
C-obstacle (distance: 328 mm)



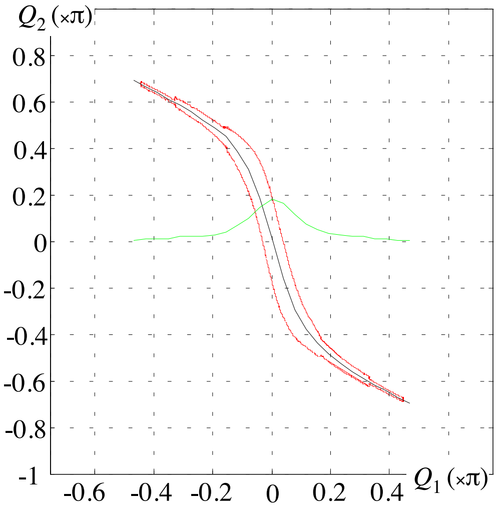
C-obstacle (distance: 308 mm)



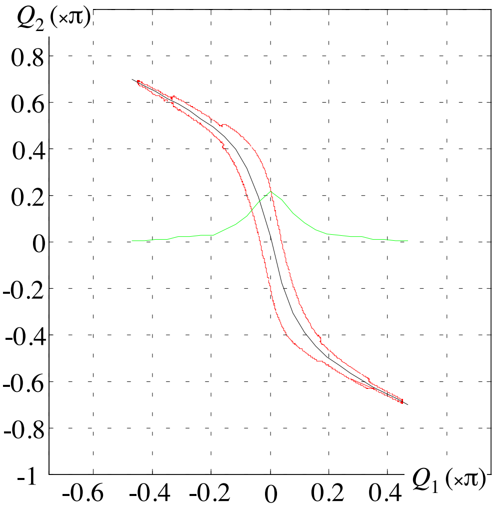
C-obstacle (distance: 298 mm)



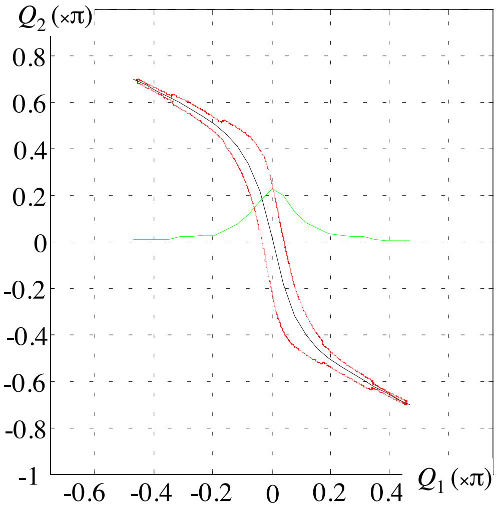
C-obstacle (distance: 288 mm)



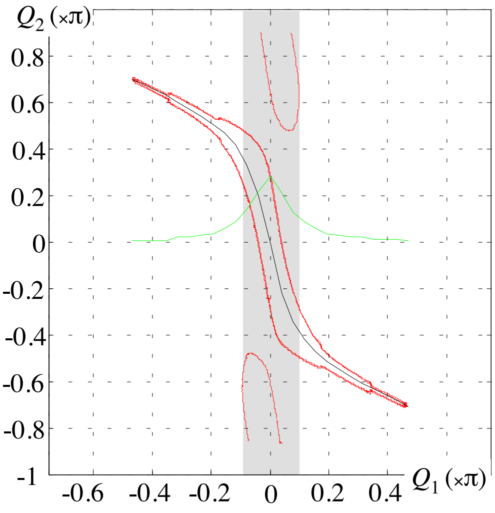
C-obstacle (distance: 283 mm)



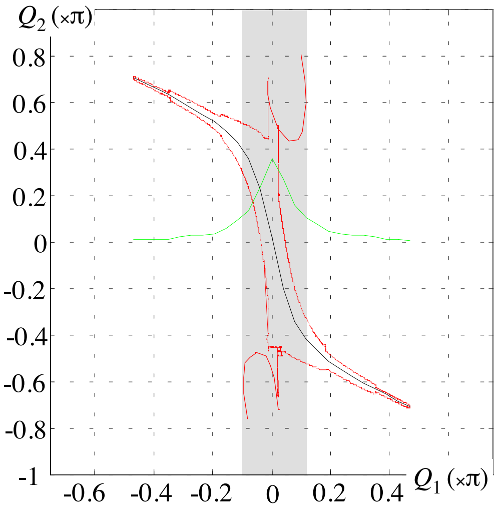
C-obstacle (distance: 278 mm)



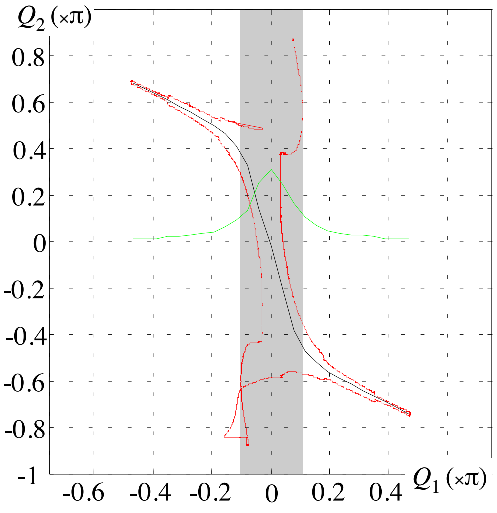
C-obstacle (distance: 273 mm)



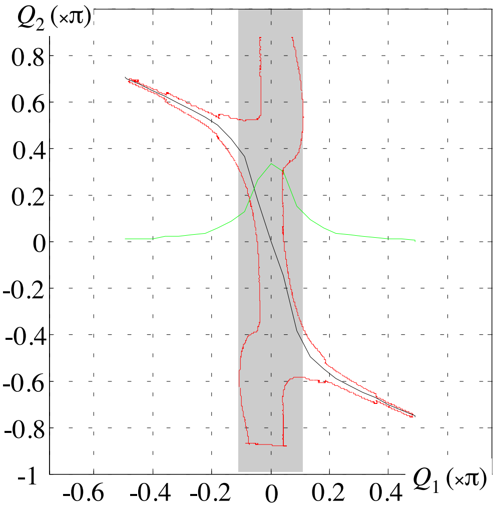
C-obstacle (distance: 268 mm)



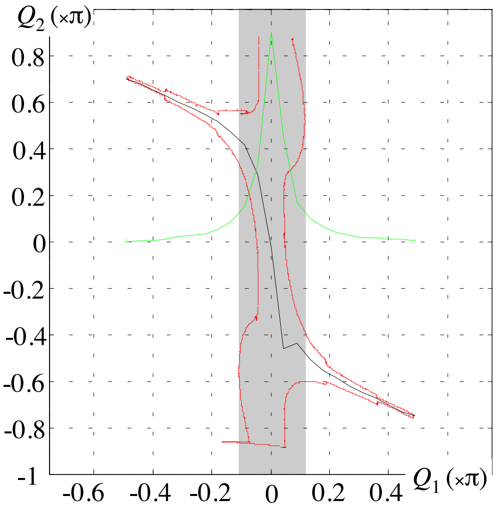
C-obstacle (distance: 263 mm)



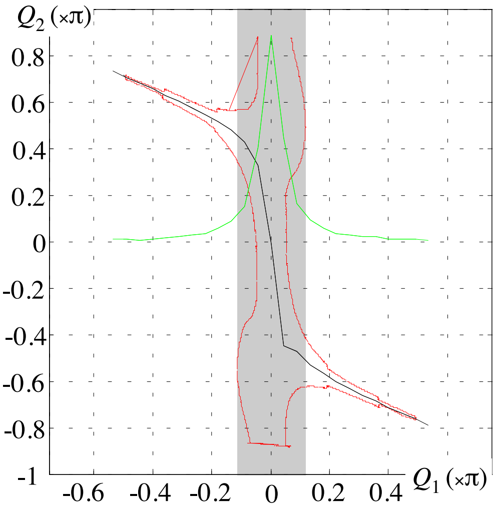
C-obstacle (distance: 258 mm)



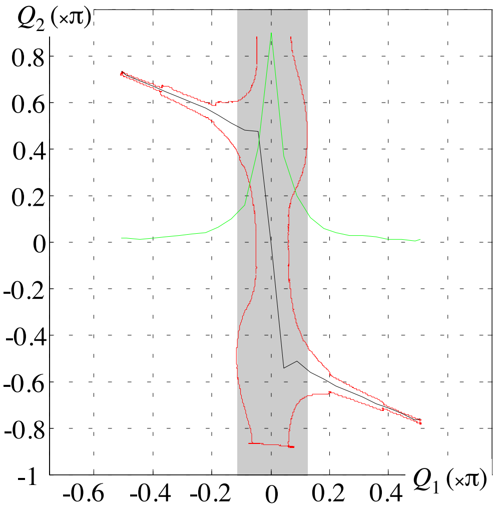
C-obstacle (distance: 253 mm)



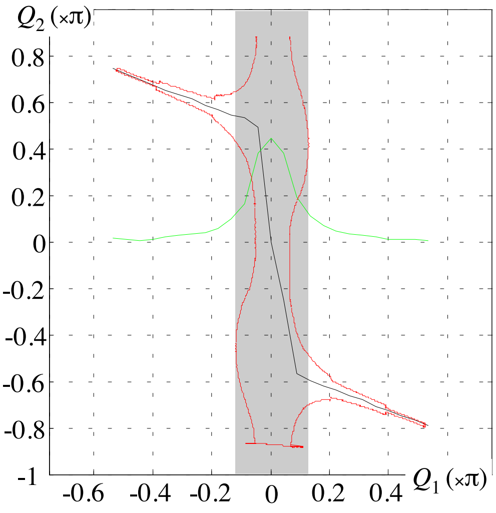
C-obstacle (distance: 248 mm)



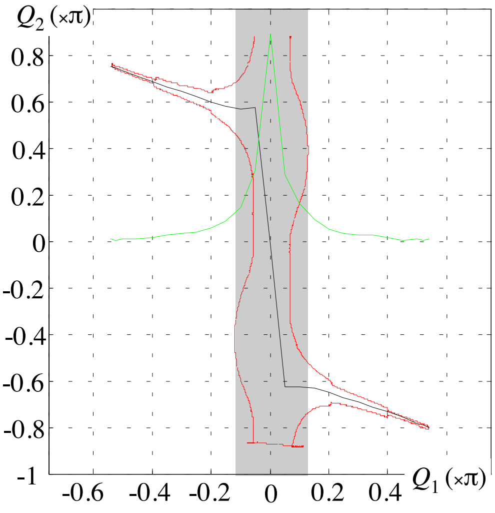
C-obstacle (distance: 238 mm)



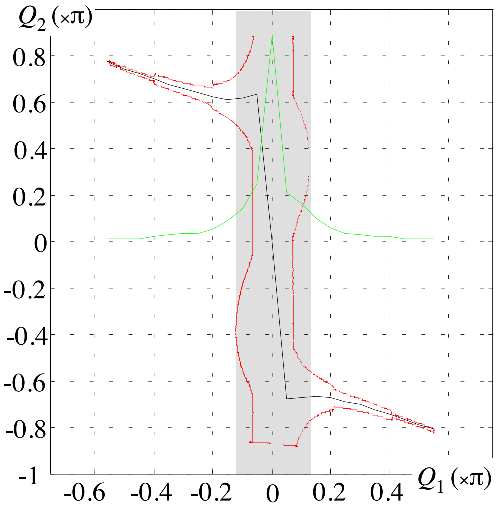
C-obstacle (distance: 228 mm)



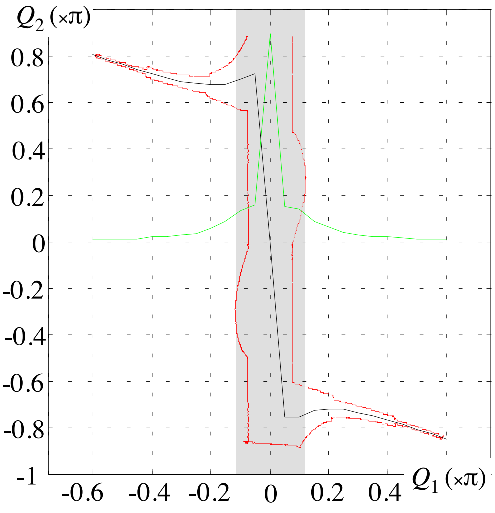
C-obstacle (distance: 218 mm)



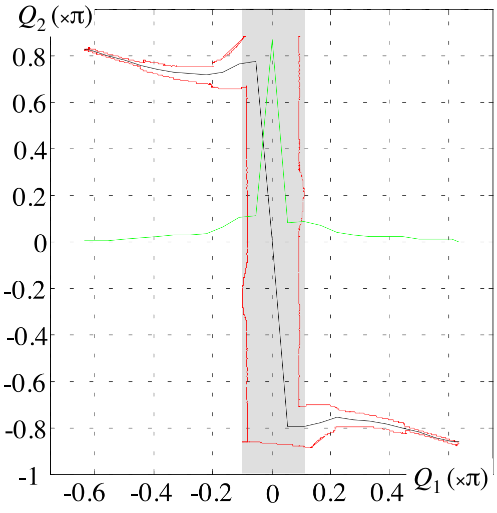
C-obstacle (distance: 208 mm)



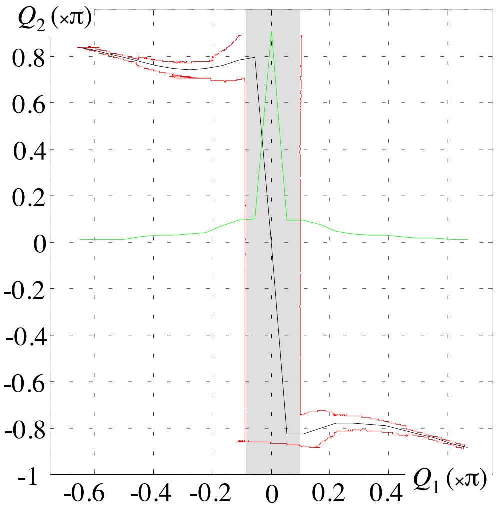
C-obstacle (distance: 188 mm)



C-obstacle (distance: 168 mm)



C-obstacle (distance: 158 mm)

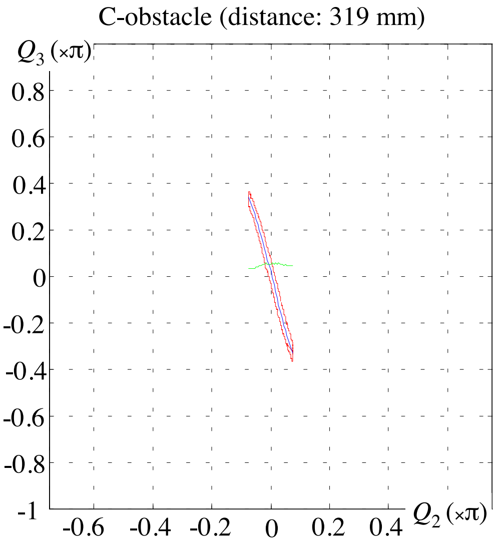


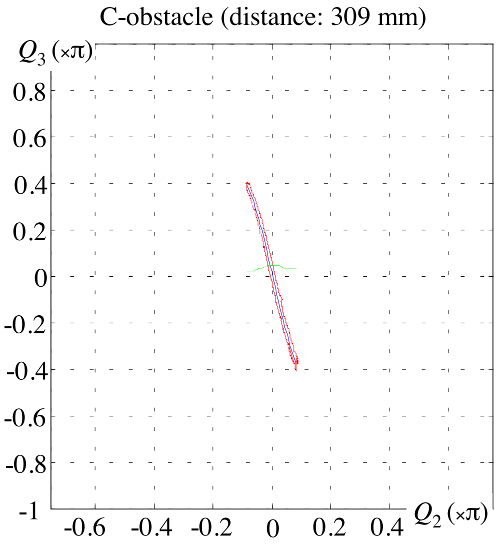
Appendix A-3

The following figures show C-patterns due to collisions between an obstacle rod and the perimeter of the elbow link (l_2) and the finger (l_3) of the MA2000-manipulator at increments of distance *d*. The shoulder link has not been considered in these measurements. The measurements have been carried out by moving the two active links of the MA2000-manipulator around the obstacle ensuring that at all times some point of the manipulator was in contact with the obstacle.

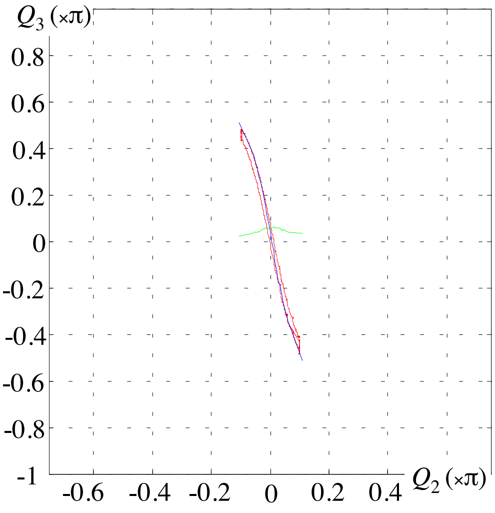
The obstacle rod has a diameter of 3 *mm*. The distance between the centre of the axis of joint g_2 and the centre of the rod is shown in the header of each figure. For obstacles in the range *d*=[269 *mm*, 319 *mm*], there are collisions with link l_3 only. For values of *d*=[229 *mm*, 259 *mm*] there are collisions with link l_2 and l_3. For values of *d*=[49 *mm*, 199 *mm*] there are collisions with link l_2 only.

Each figure shows the recorded C-space pattern, the centre line and the distance in vertical direction between the centre line and the recorded C-space pattern (dotted). The grey-shaded blocks mark those areas which describe collisions due to link l_2.

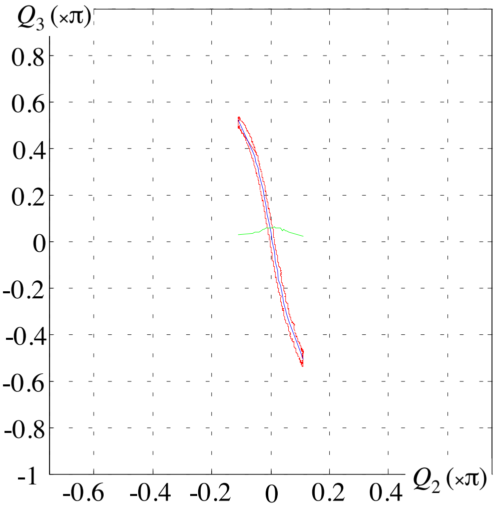

C-obstacle (distance: 319 mm)


C-obstacle (distance: 309 mm)

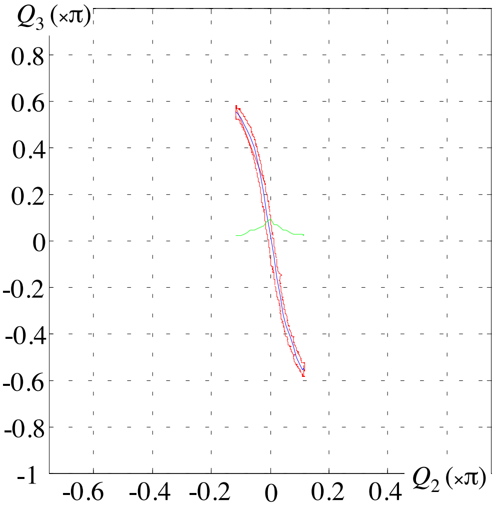
C-obstacle (distance: 299 mm)



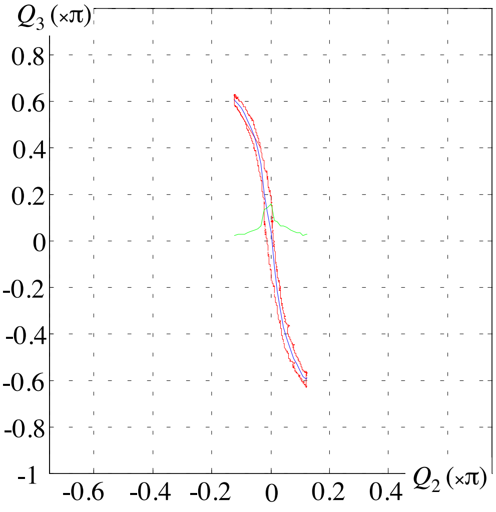
C-obstacle (distance: 289 mm)



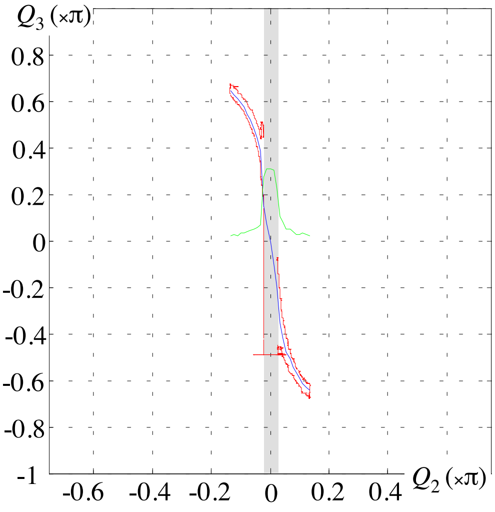
C-obstacle (distance: 279 mm)



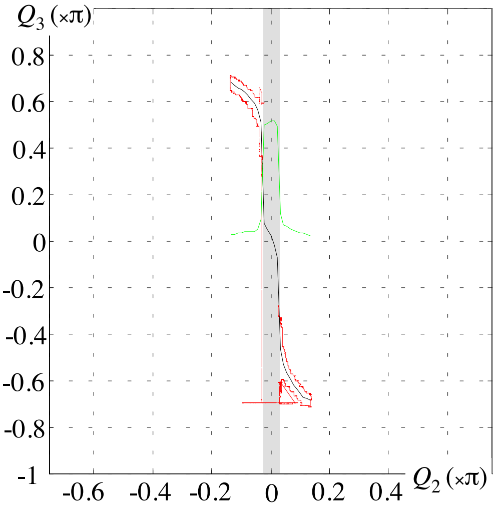
C-obstacle (distance: 269 mm)



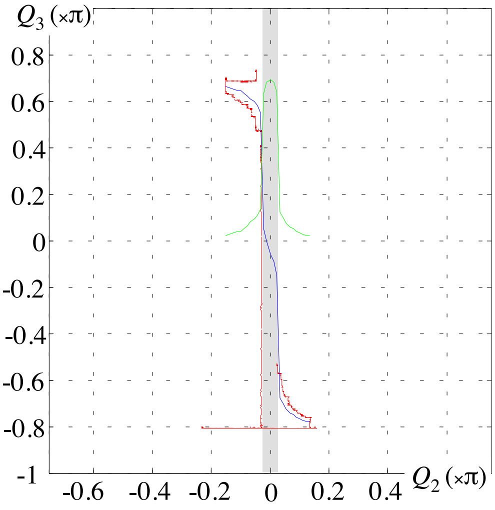
C-obstacle (distance: 259 mm)



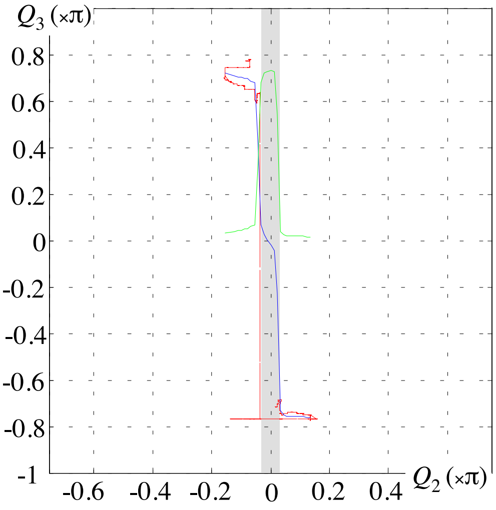
C-obstacle (distance: 249 mm)



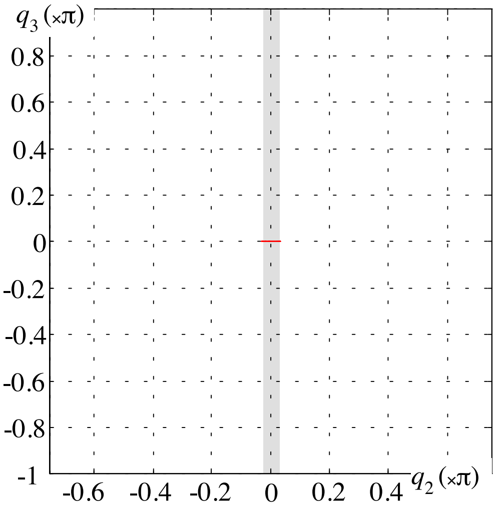
C-obstacle (distance: 239 mm)



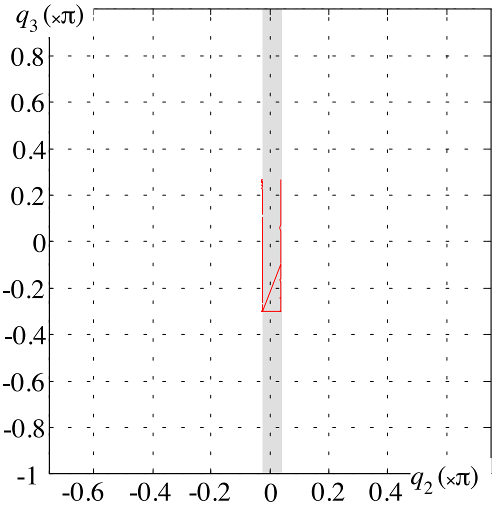
C-obstacle (distance: 229 mm)



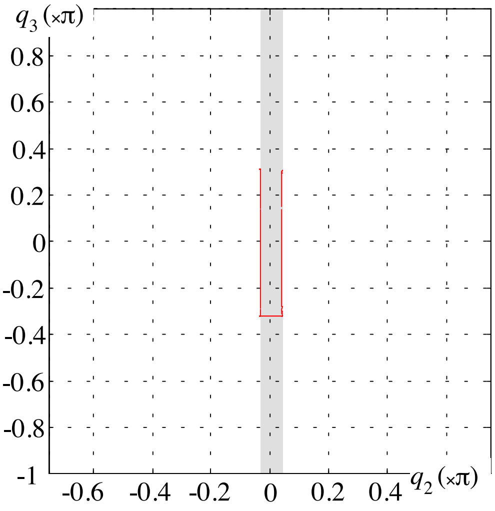
C-obstacle (distance: 199 mm)



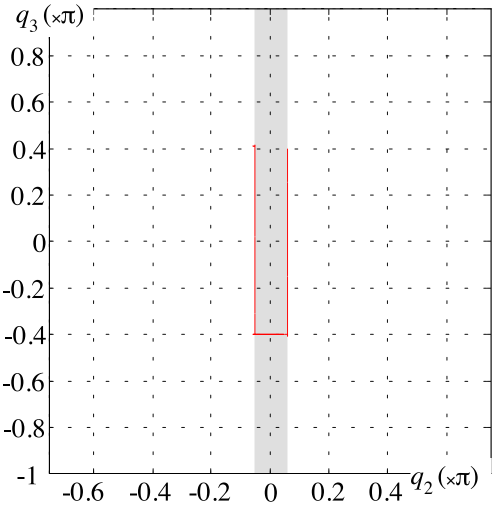
C-obstacle (distance: 179 mm)



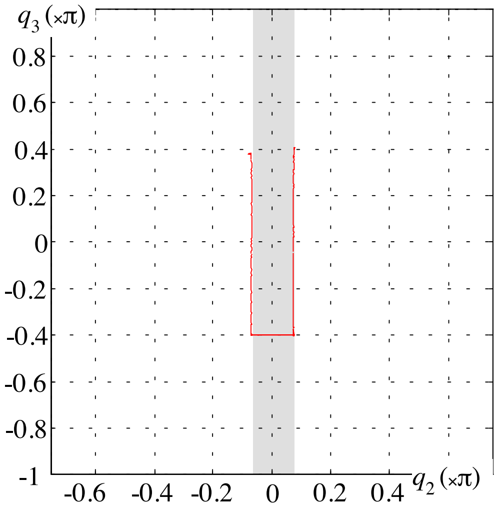
C-obstacle (distance: 159 mm)



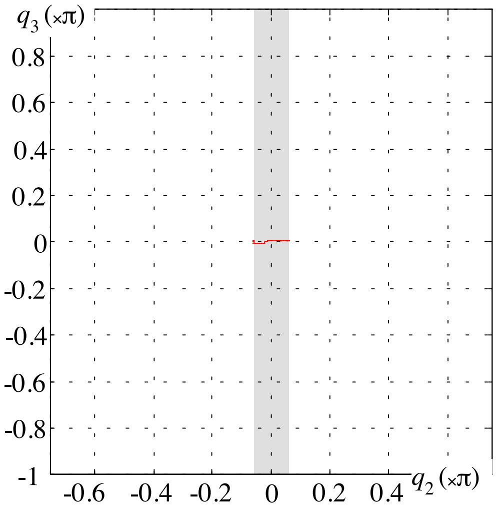
C-obstacle (distance: 139 mm)



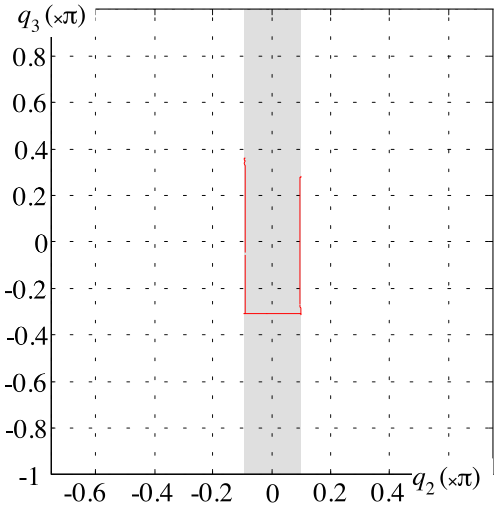
C-obstacle (distance: 119 mm)



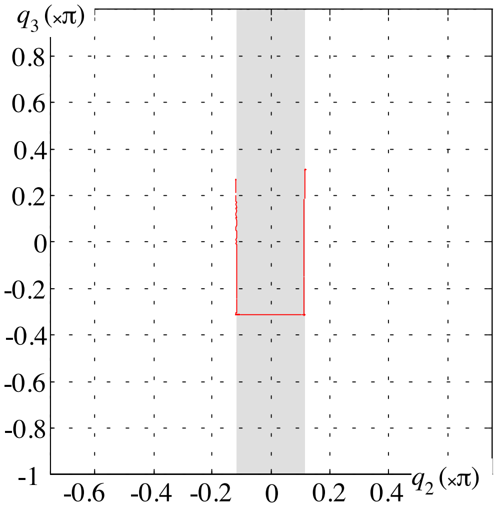
C-obstacle (distance: 109 mm)



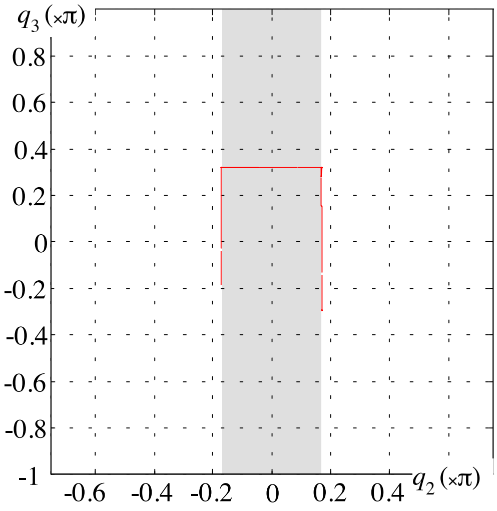
C-obstacle (distance: 99 mm)



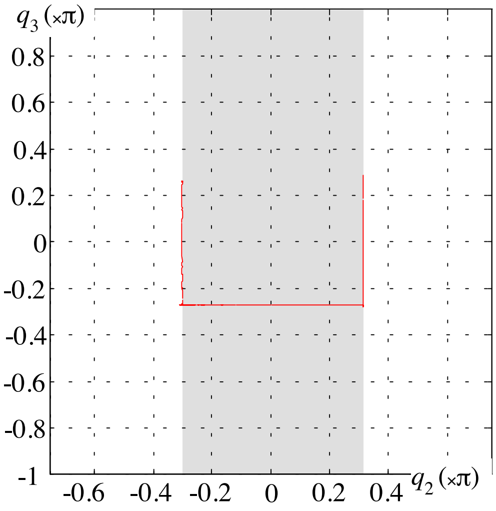
C-obstacle (distance: 89 mm)



C-obstacle (distance: 69 mm)



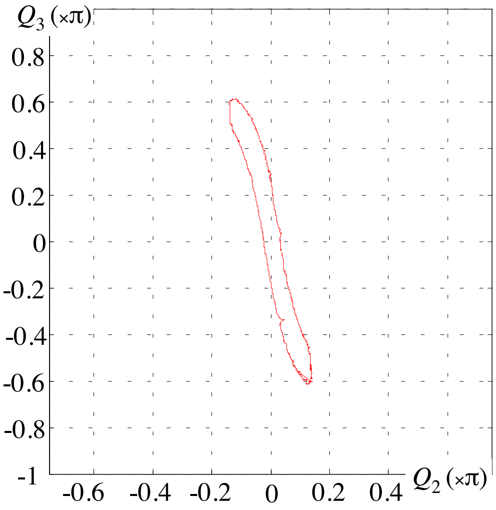
C-obstacle (distance: 49 mm)



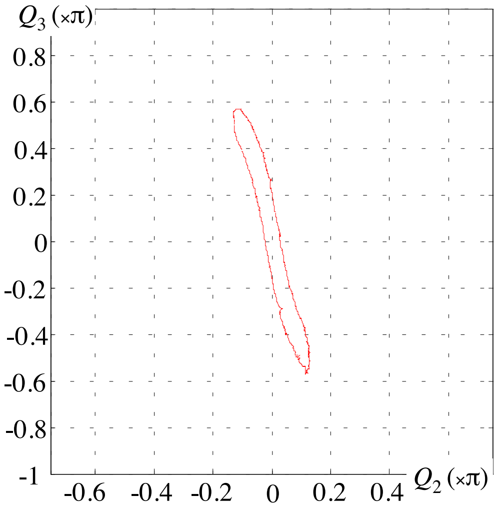
The following figures show C-obstacles due to collisions between a circular obstacle and the perimeter of link l_3 of the MA2000-manipulator at increments of distance d. Those recordings have been carried out to show that in principle it is possible to record and store the C-space patterns of more extended obstacles. As described in earlier sections link l_3 has been moved around the obstacle keeping contact with it at all times.

The obstacle has a diameter of 37 *mm*. The distance between the centre of the axis of joint g_2 and the centre of the obstacle is shown in the header of each figure.

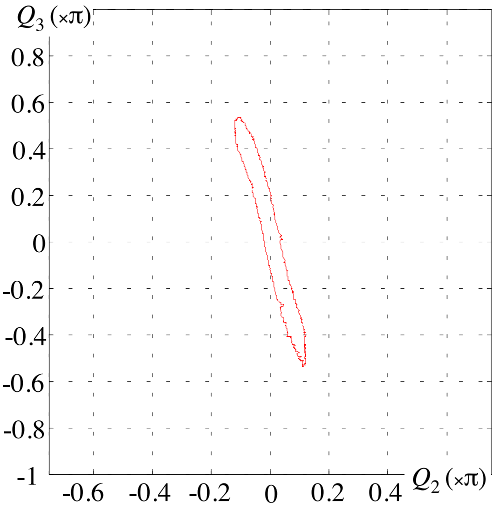
C-obstacle of disk (distance: 289 mm)



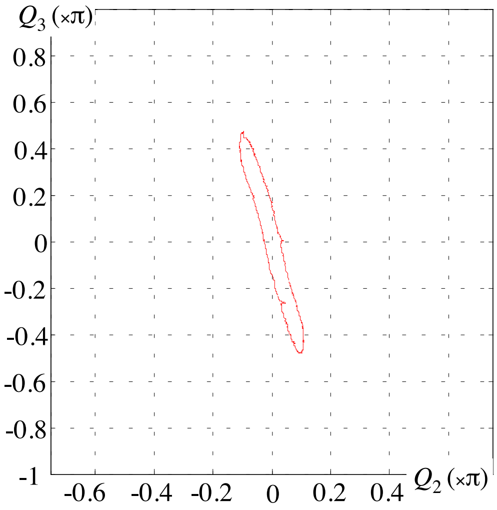
C-obstacle of disk (distance: 299 mm)



C-obstacle of disk (distance: 309 mm)



C-obstacle of disk (distance: 319 mm)



Appendix B

Convergence in the Resistive Grid

Laplace's equation describes how physical phenomena like for example the electric field and current flow behave in a continuous medium. Harmonic functions are solutions to Laplace's equation. Given appropriate boundary conditions, these harmonic functions produce a distribution surface which has only one global fixpoint (extremum). Since a resistive grid represents a discrete version of the continuous medium, the distribution after convergence also has only one global fixpoint.

The question remains whether a neuro-resistive grid or a similar neural network type emulated on a digital computer will always converge to one solution. This mainly depends on the slope of the activation function in each network node. Local minima or oscillations might occur while iterating the network. It will be shown, here, for which slopes in the activation function the network evolves into a stable activity distribution.

The following section will focus on the time-discrete type, since all of the experiments and simulations during this research have been carried out on a serial computer system.

The update algorithm for the evolution for the time-discrete Hopfield network as well as the computer emulated resistive grid can be implemented as a procedure based on the relaxation method which allows to solve a set of nonlinear equations in an iterative fashion. Here, the update formula is given in the matrix form:

$$\mathbf{u}^{(k+1)} = f\left(\mathbf{w}\mathbf{u}^{(k)} - \mathbf{I}\right), \tag{B-1}$$

where f is the activation function, and

$\mathbf{u} = \left[u_1, u_2, \ldots, u_N\right]$,

$\mathbf{u}^{(k)} = \mathbf{u}(k\tau)$ with sampling period τ,

$\mathbf{w} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & \cdots & w_{NN} \end{bmatrix}$ represents the interconnection matrix and

$\mathbf{I} = \left[I_1, I_2, \ldots, I_N\right]$ represents the external input.

Activation function f is a monotonically increasing function which is restricted to the range [0,1]. The interconnection matrix **w** defines the weight values between nodes. Those nodes which are not connected to each other are described by a zero weight value in matrix **w**. For further details regarding the update rule, see Chapter 3.

Global stability of a system which is defined by differential or difference equations can be proven using a Lyapunov function (see [Kosko92]). The Lyapunov function is a scalar which describes the entire system's energy at any time. Global stability means that the system converges to a stable equilibrium. Hopfield showed that the stable states of the system described in Eq. (B-1) are the local minima of the following Lyapunov function [Cichocki94 and references therein]:

$$L(\mathbf{u}) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}u_j u_i - \sum_{i=1}^{N} u_i I_i + \sum_{i=1}^{N}\int_0^{u_i} f^{-1}(x)dx\,. \tag{B-2}$$

The value of the integral depends on the specific shape of the activation function. To see whether the Lyapunov function has a local minimum (sufficient condition) the change of the Lyapunov function has to be calculated. The activity of each node changes from u_i to $u_i^{(k+1)} = u_i^{(k)} + \Delta u_i$ at some time $t = (k+1)\tau$ [Cichocki94]:

$$\Delta L(\mathbf{u}) = L(\mathbf{u}^{(k+1)}) - L(\mathbf{u}^{(k)})\,. \tag{B-3}$$

Thus:

$$\begin{aligned}\Delta L(\mathbf{u}) &= -\frac{1}{2}\sum_{i,j}^{N} w_{ij}(u_i^{(k+1)}u_j^{(k+1)} - u_i^{(k)}u_j^{(k)}) - \sum_{i}^{N} I_i(u_i^{(k+1)} - u_i^{(k)}) + \sum_{i}^{N}\int_{u_i^{(k)}}^{u_i^{(k+1)}} f^{-1}(x)dx \\ &\underset{(w_{ij}=w_{ji})}{=} -\frac{1}{2}\sum_{i,j}^{N} w_{ij}(2\Delta u_i u_j^{(k)} + \Delta u_i \Delta u_j) - \sum_{i}^{N} I_i \Delta u_i + \sum_{i}^{N}\int_{u_i^{(k)}}^{u_i^{(k+1)}} f^{-1}(x)dx\,.\end{aligned} \tag{B-4}$$

The integral in Eq. (B-4) can be expanded in Taylor series about $u_i^{(k+1)}$:

$$\begin{aligned}\int_0^{u_i^{(k)}} f^{-1}(x)dx = &\int_0^{u_i^{(k+1)}} f^{-1}(x)dx \\ &+\int_0^{u_i^{(k+1)}} \left(f^{-1}(x)\right)' dx \cdot (u_i^{(k)} - u_i^{(k+1)}) \\ &+\frac{1}{2}\int_0^{\xi} \left(f^{-1}(x)\right)'' dx \cdot (u_i^{(k)} - u_i^{(k+1)})^2\end{aligned} \tag{B-5}$$

with $\xi \in [u_i^{(k)}, u_i^{(k+1)}]$.

Rearranging Eq. (B-5), one gets:

$$\begin{aligned}\int_{u_i^{(k)}}^{u_i^{(k+1)}} f^{-1}(x)dx &= \left[f^{-1}(x)\right]_0^{u_i^{(k+1)}} \Delta u_i - \frac{1}{2}\left[(f^{-1})'(x)\right]_0^{\xi} \Delta u_i^2 \\ &= f^{-1}(u_i^{(k+1)})\Delta u_i - \frac{1}{2}(f^{-1})'(\xi)\Delta u_i^2\,.\end{aligned} \tag{B-6}$$

Inserting Eq. (B-6) in Eq. (B-4) and using Eq. (B-1), one obtains

$$\begin{aligned}\Delta L &= -\sum_{i}^{N} \Delta u_i \sum_{j}^{N} (w_{ij} u_j{}^{(k)} + I_i) - \frac{1}{2}\sum_{i,j}^{N} w_{ij}\Delta u_i \Delta u_j \\ &\quad + \sum_{i}^{N} [f^{-1}(u_i{}^{(k+1)})\Delta u_i - \frac{1}{2}(f^{-1})'(\xi)\Delta u_i{}^2] \\ &= \underbrace{-\sum_{i}^{N} \Delta u_i \sum_{j}^{N} [\underbrace{(w_{ij} u_j{}^{(k)} + I_i)}_{f^{-1}(u_i{}^{(k+1)})} - f^{-1}(u_i{}^{(k+1)})]}_{=0} \\ &\quad - \frac{1}{2}\sum_{i,j}^{N} w_{ij}\Delta u_i \Delta u_j - \sum_{i}^{N} \frac{1}{2}(f^{-1})'(\xi)\Delta u_i{}^2 \\ &= -\frac{1}{2}\sum_{i,j}^{N} [w_{ij} + \delta_{ij}(f^{-1})'(\xi)]\Delta u_i \Delta u_j \\ &= -\frac{1}{2}\mathbf{u}^{\mathbf{T}}[\mathbf{w} + (f^{-1})'(\xi)\mathbf{E}]\mathbf{u}\end{aligned} \tag{B-7}$$

with δ_{ij} being the Kronecker delta-function and **E** being the identity matrix.

If all the eigenvalues of a symmetric matrix are strictly positive, the matrix is positive definite. Thus, ΔL is negative if matrix $\mathbf{w} + (f^{-1})'(\xi)\mathbf{E}$ is positive definite. The eigenvalues of matrix $\mathbf{w} + (f^{-1})'(\xi)\mathbf{E}$ can be calculated as follows:

$$\det(\mathbf{w} + (f^{-1})'(\xi)\mathbf{E} - \lambda\mathbf{E}) = 0,$$

where λ_i are the eigenvalues of **w**.

Matrix $\mathbf{w} + (f^{-1})'(\xi)\mathbf{E}$ is positive definite, if all its eigenvalues are positive:

$$(f^{-1})'(\xi) - \lambda_i > 0 \;\Leftrightarrow\; \lambda_i < (f^{-1})'(\xi) = \frac{1}{f'(x)},$$

where $\xi = f(x)$.

Thus, the following condition has to be satisfied:

$$\frac{1}{\lambda_{min}} > \max(f'(x)). \tag{B-8}$$

In the case that the activation function is linear, $f(x) = kx$, the first derivative $f'(x)$ is the constant k and the reciprocal of the most negative eigenvalue of **w** has to be greater than this constant k.

For the activation functions often used in Hopfield networks (tanh, linear) an estimation can be made for $(f^{-1})'(\xi)$ which ensures that $\mathbf{w} + (f^{-1})'(\xi)\mathbf{E}$ is positive definite. If $f(x)$ is a non-linear function then $1/\lambda_{\min}$ has to be greater than the maximum slope of $f(x)$. In case, the activation function is $f(x) = \tanh(kx)$, it follows that $f'(x) = \dfrac{k}{\cosh^2(kx)}$ which has its maximum at $f'_{\max}(x_0 = 0) = k$.

The preceding paragraphs showed that the described resistive grid/Hopfield network converges. To show that the found solution is unique, one has to show that the Lyapunov function is strictly convex [Glasius94]. If and only if the Hessian matrix of the Lyapunov function is positive definite then the Lyapunov function itself is strictly convex and the grid/net converges to one global solution.

Thus:

$$L_{ij} = -w_{ij} + \delta_{ij}\frac{1}{f'(f^{-1}(u_i))} \le -w_{ij} + \delta_{ij}\frac{1}{f'(\xi)} = -\mathbf{w} + \frac{\mathbf{1}}{f'(\xi)}\mathbf{E}, \qquad \text{(B-9)}$$

and for the eigenvalues follows:

$$\frac{1}{\lambda_{max}} > \max(f'(x)), \qquad \text{(B-10)}$$

where $\lambda_{\max}$ is the most positive eigenvalue. If Eq. (B-10) is satisfied, the Lyapunov function is strictly convex.

Combining Eqs. (B-8) and (B-10), one obtains [Glasisus94]:

$$\frac{1}{\lambda} > \max(f'(x)), \quad \lambda = max\{|\lambda_{min}|, \lambda_{\max}\}. \qquad \text{(B-11)}$$

If Eq. (B-11) is satisfied, the network or grid will converge to one unique solution.

In the resistive grid, matrix **w** has only elements which are zero or positive. The non-zero elements represent the weights between neighbouring nodes. It can be shown that if there are no obstacles in the grid, all eigenvalues are positive and the largest eigenvalue is equal to the sum of the non-zero elements per row or column [Glasius94]. The largest eigenvalue of a obstacle-free grid is larger than the largest eigenvalue in a grid with obstacles. To assure that the grid converges to a stable solution, the maximum slope of the activation function *f* should be smaller than the reciprocal of the sum of the non-zero elements per row or column in the absence of obstacles (see also Chapter 3).

Bibliography

[Aiyer90] Aiyer, S.V.B., Niranjan, M., Fallside, F., “A Theoretical Investigation into the Performance of the Hopfield Model”, *IEEE Transactions on Neural Networks*, Vol. 1, No. 2, pp. 204-215, June 1990.

[Althoefer93] Althoefer, K., Fraser, D.A., Azhar, F., "Investigation of an Adaptive Neural Network using Comdisco SPW", *Proceedings of the second annual one-day seminar of industrial case studies in DSP and Communications systems Modelling*, Comdisco Systems, June 15, 1993.

[Althoefer94] Althoefer, K., “On the C-Space Transformation for Planar Manipulators with *n* links”, (not published), Dept. of Electronic & Electrical Eng., King's College, London, 1994.

[Althoefer94a] Althoefer, K., Fraser, D.A., "On-line Neural Network Training: The Backpropagation Algorithm under Comdisco SPW", *Proceedings of the third annual international seminar series*, Alta Group of CADENCE Design Systems, June 1994.

[Althoefer94b] Althoefer, K., “A Hierarchical Neural-Network-Based Path Planning Concept”, MPhil to PhD Transfer Thesis, Dept. of Electronic & Electrical Engineering, King's College, London, 1994.

[Althoefer95] Althoefer, K., Fraser, D.A., Bugmann, G., Turán, J., "The Configuration Space Transformation for Articulated Manipulators: A Novel Approach based on RBF-Networks", *Proceedings of the Fourth International Conference on 'Artificial Neural Networks'*, IEE, Cambridge, UK, pp. 245-249, June 16-28, 1995.

[Althoefer95b] Althoefer, K., Fraser, D.A., Bugmann, G., "The Configuration Space for Manipulators computed by a Basis-Function Network", *Proceedings of the International Conference on Engineering Applications of Neural Networks*, Otaniemi/Helsinki, Finland, pp. 469-472, August 21-23, 1995.

[Althoefer95c] Althoefer, K., Fraser, D.A., Bugmann, G., Plumbley, M.D., "Asymmetric B-Splines for the fast Calculation of C-Space Patterns of Robot Arms", *Proceedings of the 5th International Conference on Artificial Neural Networks (ICANN'95)*, Vol. 2, pp. 387-392, Paris, France, October 9-13, 1995.

[Althoefer95d] Althoefer, K., Bugmann, G., "Planning and Learning Goal-Directed Sequences of Robot-Arm Movements", *Proceedings of the 5th International Conference on Artificial Neural Networks (ICANN'95)*, Vol. 1, pp. 449-454, Paris, France, October 9-13, 1995.

[Althoefer95e] Althoefer, K., Fraser, D.A., Bugmann, G., "Rapid Path Planning for Robotic Manipulators using an emulated Resistive Grid", *Electronics Letters*, Vol. 31, No. 22, pp. 1960-1961, Stevenage, UK, October 26, 1995.

[Althoefer96] Althoefer, K., Fraser, D.A., “Fuzzy Obstacle Avoidance for Robotic Manipulators”, *Neural Network World*, Vol.6, No. 2, pp.131-142, 1996.

[Althoefer96c] Althoefer, K., Fraser, D.A., “Robotic Manipulators Amidst Moving Obstacles: Fuzzy-based Obstacle Avoidance”, *Proceedings of EUFIT '96 - The Fourth European Congress on Intelligent Techniques and Soft Computing*, Aachen, September 2-5, 1996.

[Azhar93] Azhar, F., Fraser, D.A., “Adaptive Robot Control using a Processor Network”, *Proceedings of EUROMICRO Workshop on Parallel and Distributed Processing*, January 29, 1993.

[Bapi95] Bapi, R., D'Cruz, B. and Bugmann, G. (1995) "Neuro-Resistive Grid Approach to Pole-Balancing Problem", *Proceedings of the International Conference on Neural Networks (ICANN'95)*, Paris, France, pp. 539-544, Oct. 9-13, 1995.

[Barraquand91] Barraquand, J., Latombe, J.-C., “Robot Motion Planning: A Distributed Representation Approach”, *The International Journal of Robotics Research*, Vol. 10, No. 6, pp. 628-649, December 1991.

[Barraquand92] Barraquand, J., Latombe, J.-C., “Numerical Potential Field Techniques for Robot Path Planning”, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 22, No. 2, pp. 222-241, March/April 1992.

[Beiu96] Beiu, V., “Entropy bounds for classification algorithms”, *Neural Network World*, Vol. 6, No. 4, pp. 497-505, 1996.

[Bellman57] Bellman, R.E., *Dynamic Programming*, Princeton University Press, Princeton, 1957.

[Bellman62] Bellman, R.E., Dreyfus, S.E., *Applied Dynamic Programming*, Princeton University Press, Princeton, 1962.

[Berenji92] Berenji, H.R., Khedkar, P., “Learning and Tuning Fuzzy Logic Controllers Through Reinforcements”, *IEEE Transactions on Neural Networks*, Vol. 3, No. 5, pp. 724-740, September 1992.

[Bessiere93] Bessière, P., Ahuactzin, J.M., Talbi, E.G., Mazer, E., “The ‘Ariadne’s Clew’ Algorithm: Global Planning with Local Methods”, *Proceedings of IEEE Conference on Intelligent Robots and Systems (IROS)*, Yokohama, Japan, 1993.

[Bishop95] Bishop, C.M., *Neural Networks for Pattern Recognition*, Oxford University Press, UK, 1995.

[Bose96] Bose, N.K., Liang, P., *Neural Network Fundamentals with Graphs, Algorithms, and Applications*, McGraw-Hill, Inc., 1996.

[Boult90] Boult, T.E., “Dynamic Digital Distance Maps in Two Dimensions”, *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 5, October 1995.

[Branicky90] Branicky, M.S., Newman, W.S., “Rapid Computation of Configuration Space Obstacles”, *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pp. 304-310, 1990.

[Bronshtein85] Bronshtein, I.N., Semendyayew, K.A., *Handbook of Mathematics*, Van Nostrand Reinhold Company, New York, USA, 1985.

[Brooks86] Brooks, R.A., “A Robust Layered Control System For A Mobile Robot”, *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, pp. 14-23, March 1986.

[Brown94] M. Brown, C. Harris, *Neurofuzzy Adaptive Modelling and Control*, Prentice Hall International Ltd., UK, 1994.

[Bugmann94] Bugmann, G., Denham, M.J., Taylor, J.G., “Sensor and memory based path planning in the egocentric reference frame of an autonomous mobile robot”, Internal Report (NRG-94-01), School of Computing, University of Plymouth, UK, 1994.

[Bugmann95] Bugmann, G., Taylor, J.G., Denham, M.J., “Route finding by neural nets”, *Neural Networks*, Taylor, J.G. (editor), Alfred Waller Ltd., Henley-on-Thames, UK, pp. 217-230, 1995.

[Bugmann96] Bugmann, G., "Value Maps for planning and learning implemented with cellular automata ", *Proceedings of the 2nd International Conference on Adaptive Computing in Engineering Design and Control (ACEDC'96)*, Parmee I.C. (ed.), pp. 307-309, Plymouth, 26-28 March, 1996.

[Chen92] Chen, N., Chung, H., “Robot Path Planner: A Neural Networks Approach”, *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Raleigh, NC, USA, pp. 548-553, July 7-10, 1992.

[Chien95] Chien, Y.-P., Xue, Q., Chen, Y., “Configuration Space Model of Tightly Coordinated Two Robot Manipulators Operating in 3-Dimensional Workspace”, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 25, No. 4, April 1995.

[Cichocki94] Cichocki, A., Unbehauen, R., *Neural Networks for Opimization and Signal Processing*, John Wiley & Sons Ltd. & B.G. Teubner, Stuttgart, Germany, 1994.

[Cires96] Cires, J., Zufiria, P.J., “The Self-organizing Map as a Perceptual Level for Mobil Robot Control”, *Proceedings of the 2nd International Conference on Engineering Applications of Neural Networks*, pp. 183-187, London, UK, 17-19 June, 1996.

[Connolly90] Connolly, C.I., Burns, J.B., Weiss, R., “Path Planning Using Laplace’s Equation”, *Proceedings of the IEEE Conference on Robotics and Automation*, Los Alamos, USA, pp. 2102-2106, 1990.

[Connolly93] Connolly, C.,I., Burns, J.B., “A model for the functioning of the striatum”, *Biological Cybernetics*, 68, pp. 535-544, Springer-Verlag, 1993.

[Coolen91] Coolen, A.C.C., Kröse, B., Noest, A.J., “Neural Robot Vision and Control”, Project proposal commissioned by the Dutch Foundation for Neural Networks, Nijmegen, 1991.

[Cooper81] Cooper, L., Cooper, M.W., *Introduction to Dynamic Programming*, Pergamon Press Ltd., 1981

[Cox92] Cox, E., “Fuzzy fundamentals”, *IEEE Spectrum*, pp. 58-61, October 1992.

[D'Cruz96] D'Cruz B., Bapi, R., Bugmann, G., "Neuro-resistive grid approach to trainable controllers: A pole balancing example", *Neural Computing and Application Journal*, 1996 (forthcoming)

[Drews92] Drews P., Wehninck, F.J.S., Strunz, U., Willms, K., “A sensor system for beam tracking and geometry detection for arc-welding”, *Robotersysteme*, Vol.8, No.3, pp.148-154, 1992 (in German).

[Droesser94] Drösser, C., *Fuzzy Logic: Methodische Einführung in krauses Denken*, Rowohlt Taschenbuch Verlag GmbH, Hamburg, Germany, 1994 (in German).

[El-Mousa96] El-Mousa, A.H., Clarkson, T.G., “Multi-configurable pram based neuro-computer”, *Neural Network World*, Vol. 6, No. 4, pp. 587-596, 1996.

[Fox92] Fox, J.J., Maciejewski, A.A., "Computing the Topology of Configuration Space", *Proceedings of the 1992 IEEE International Conference System, Man, Cybernetics*, Chicago, IL, pp. 31-36, October 1992.

[Fox94] Fox , J.J., Maciejewski, A.A., "Utilizing the Topology of Configuration Space Real-time Multiple Manipulator Path Planning," *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS '94)*, Munich, Germany, September 12-16, 1994.

[Fraser93] Fraser, D.A., Azhar, F., Althoefer, K., "On-line Adaptation in Robot Control", *Proceedings of the Third International Conference on 'Artificial Neural Networks'*, IEE, Brighton, UK, pp. 205-209, May 25-27, 1993.

[Freund96] Freund, E., Dierks, F., "Map-based free navigation for autonomous vehicles", *International Journal of Systems Science*, Vol. 27, No. 8, pp.753-770, 1996.

[Glasius94] Glasius, R., Komoda, A., Gielen, S., "Neural network dynamics for path planning and obstacle avoidance", *Neural Networks*, Vol. 8., No. 1, pp. 125-133, Elsevier Science Ltd., USA, March 1994.

[Gouzenes84] Gouzènes, L., "Strategies for Solving Collision-Free Trajectories Problems for Mobile and Manipulator Robots", *International Journal of Robotics Research*, Vol. 3, No. 4, pp. 51-65, 1984.

[Graf88] Graf, D.H., LaLonde, W.R., "A Neural Controller for Collision-Free Movement of General Robot Manipulators", *Proceedings of the 1988 IEEE International Conference on Neural Networks*, San Diego, California, USA, pp. I-77-84, July 24-27, 1988.

[Graham90] Graham, I., King, T., *The Transputer Handbook*, Prentice Hall International Ltd., 1990.

[Gullapalli94] Gullapalli, V., Franklin, J., Benbrahim, H., "Acquiring Robot Skills via Reinforcement Learning", *IEEE Control Systems*, pp. 13-24, February 1994.

[Gupta90] Gupta, K.K., "Fast Collision Avoidance for Manipulator Arms: A Sequential Search Strategy", *IEEE Transaction on Robotics and Automation*, Vol. 6, No.5, October 1990.

[Gupta92] Gupta, K.K., Guo, Z., "Motion Planning for Many Degrees of Freedom: Backtracking with Sequential Search", *Proceedings of IEEE Conference on Robotics and Automation*, 1992.

[Handelman90] Handelman, D.A., Lane, S.H., Gelfand, J.J., "Integrating Neural Networks and Knowledge-Based Systems for Intelligent Robotic Control", *IEEE Control Systems Magazine*, April 1990.

[Helliwell95] Helliwell, I.S., Turega, M. A., Cottis, R. A., "Accountability of Neural Networks Trained with 'Real World' Data", in *Proceedings of the International Conference on 'Artificial Neural Networks'*, pp. 218-222, Cambridge, 26-28 June, 1995.

[Higgins88] Higgins, T.C., "Slave/Stand-Alone Transputer Board", Masters' Thesis at the Dept. of Electronic and Electrical Eng., King's College London, 1988.

[Higgins94] Higgins, C.M., Goodman, R.M., "Fuzzy Rule-Based Networks for Control", *IEEE Transactions on Fuzzy Systems,* Vol. 2, No. 1, February 1994.

[Hockney88] Hockney, R.W., Eastwood J.W., *Computer simulation using particles*, Special Student Edition, 1988.

[Hoffmann96] Hoffmann, F., Malki, O., Pfister, G., "Evolutionary Algorithms for Learning of Mobile Robot Controllers", *Proceedings of EUFIT '96 - The Fourth European Congress on Intelligent Techniques and Soft Computing*, Aachen, September 2-5, 1996.

[Hush93] Hush D.R., Horne, B.G., "Progress in Supervised Networks - What's New Since Lippmann?", *IEEE Signal Processing Magazine*, January 1993.

[Hwang90] Hwang, Y.K., "Boundary Equations of Configuration Obstacles for Manipulators", *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 298-303, 1991.

[Indyk94] Indyk, D., Velastin, S.A., "Survey of Range Vision Systems", *Mechatronics*, Vol. 4, No. 4, pp.417-449, June 1994.

[Jaitly96] Jaitly, R., Fraser, D.A., "Automated 3D Object Recognition and Library Entry System", *Neural Network World*, Vol. 6, No. 2, pp. 173-183, 1996.

[Jaitly96b] Jaitly, R., Althoefer, K., Fraser, D., "From Vision to Path Planning: A Neural Based Implementation", *Proceedings of the 2nd International Conference on Engineering Applications of Neural Networks*, pp. 209-212, London, UK, 17-19 June, 1996.

[Jaitly96c] Jaitly, R., "Model-based 3D Object Recognition" (provisional title), PhD Thesis at the Dept. of Electronic and Electrical Eng., King's College London, 1996 (*forthcoming*).

[Jang92] Jang, J-S.R., "Self-Learning Fuzzy Controllers Based on Temporal Back Propagation", *IEEE Transactions on Neural Networks*, Vol. 3, No. 5, pp. 714-723, September 1992.

[Jiang94] Jiang, K., "Planning Collision-Free Operations for Mobile Robots and Manipulators", PhD-Thesis, King's College London, 1994.

[Johnson93] Johnson, J. H., Picton, P. D., Hallam, N. J., "Safety-critical neural computing: explanation and verification in knowledge augmented neural networks", *Artificial Intelligence in Engineering*, Vol. 8, pp. 307-313, Elsevier Science Limited, 1993.

[Kanaya94] Kanaya, M., Tanaka, M., "Robot Multi-Driving Controls by Cellular Neural Networks", *Proceedings of the Third IEEE International Workshop on Cellular Neural Networks and their Applications*, Rome, Italy, pp. 481-486, December 18-21, 1994.

[Kartalopoulos96] Kartalopoulos, S.V., "Tutorial in application of fuzzy logic & neural networks in communications", *Neural Network World*, Vol. 6, No. 4, pp. 417-445, 1996.

[Kassim95] Kassim, A.A., Kumar, B.V.K.V., "Potential Fields and Neural Networks", *The Handbook of Brain Theory and Neural Networks*, Arbib, M.A. (ed.), MIT Press, pp. 749-753, 1995.

[Keerthi95] Keerthi, S.S., Ravindran, B., "A Tutorial Survey of Reinforcement Learning", Technical Report, Dept. of Computer Science and Automation, Indian Institute of Science, Bangalore, 1995.

[Kelly90] Kelly, D.G., "Stability in Contractive Nonlinear Neural Networks", *IEEE Transactions on Biomedical Engineering*, Vol. 37, No. 3, pp. 231-242, March 1990.

[Khatib85] Khatib, O., “Real-time obstacle avoidance for manipulators and mobile robots”, *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 500-506, March 1985.

[Khatib86] Khatib, O., “Real-time obstacle avoidance for manipulators and mobile robots”, *The International Journal of Robotics Research*, Vol. 5, No. 1, pp. 90-98, Spring 1986.

[Kim91] Kim, J.-O., Khosla, P., “Real-Time Obstacle Avoidance Using Harmonic Potential Functions”, *Proceedings of the IEEE International Conference on Robotics and Automation*, Sacramento, California, USA, pp. 790-796, April 1991.

[Klawsky93] Klawsky, R.S., *The Science of Virtual Reality and Virtual Environments: A Technical, Scientific and Engineering Reference on Virtual Environments*, Addison-Wesley Publishing Company, 1993.

[Koch96] Koch, C., Mathur, B., “Neuromorphic vision chips”, *IEEE Spectrum*, pp. 38-46, May 1996.

[Kosko92] Kosko, B., *Neural Networks and Fuzzy Systems*, Prentice-Hall International Editions, 1992.

[Kramer96] Kramer, A. H., “Array-Based Analog Computations: Principles, Advantages and Limitations”, *Proceedings of MicroNeuro'96*, Lausanne, Switzerland, IEEE Computer Society Press, Los Alamitos, CA, pp. 68-79, 12-14 February, 1996.

[Kuperstein91] Kuperstein, M., “INFANT Neural Controller for Adaptive Sensory-Motor Coordination”, *Neural Networks*, Vol. 4, pp. 131-145, Pergamon Press plc., 1991.

[Latombe91] Latombe, J.C., *Robot Motion Planning*, Kluwer Academic, Boston, MA, USA, 1991.

[Lee96] Lee, D., *The Map-Building and Exploration Strategies of a Simple Sonar-Equipped Mobile Robot: An Experimental, Quantitative Evaluation*, Cambridge University Press, UK, 1996.

[Lozano79] Lozano-Pérez, T. and Wesley, M.A., “An Algorithm for planning Collision-free Paths among Polyhedral Obstacles”, *Communications of the ACM*, 22(10), pp. 560-570, 1979.

[Lozano83] Lozano-Peréz, T., “Spatial Planning: A Configuration Space Approach”, *IEEE Transactions on Computers*, Vol. C-32, No. 2, pp. 108-120, February 1983.

[Lozano87] Lozano-Peréz, T., "A Simple Motion-Planning Algorithm for General Robot Manipulators", *IEEE Journal of Robotics & Automation*, RA-3(3), pp. 224-238, June 1987.

[Maciejewski93] Maciejewski, A.A., Fox, J.J., “Path Planning and the Topology of Configuration Space”, *IEEE Transaction on Robotics and Automation*, Vol. 9, No. 4, August 1993.

[Mahadevan92] Mahadevan, S., Connell, J., “Automatic programming of behaviour-based robots using reinforcement learning”, *Artificial Intelligence*, 55, pp. 311-365, Elsevier, 1992.

[Maier95] Maier, S., Hanebeck, U.D., “Fuzzy Guidance of Omnidirectional Mobile Robots Including Sensor-Based Obstacle Avoidance”, *Proceedings of the Telerobotics Congress, Teleman - HCM*, Delft University, Netherlands, July 1995.

[Mamdani81a] Mamdani, E.H., Assilian, S., “An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller”, *Fuzzy Reasoning and its Applications*, Mamdani, E.H., Gaines, B.R. (eds.), Academic Press, New York, USA, pp. 311-323, 1981.

[Mamdani81b] Mamdani, E.H., “Advances in the linguistic synthesis of fuzzy controllers”, *Fuzzy Reasoning and its Applications*, Mamdani, E.H., Gaines, B.R., (eds.), pp. 325-342, Academic Press, 1981.

[Marshall94] Marshall, G.F., Tarassenko, L., “Robot Path Planning Using VLSI Resistive Grids”, *IEE Proceedings on Vision, Image and Signal Processing*, Vol. 141, No. 4, pp. 267-272, August 1994.

[Martelli76] Martelli, A., "An Application of Heuristic Search Methods to Edge and Contour Detection", *Communications of the ACM*, vol. 19, no. 2, pp. 73-83, 1976.

[Maties94] Maties, V., Precup, T., Precup, D., Sipos C., “Automatic simulation of robot obstacle avoidance using fuzzy algorithms, SINTES 7, *The VII-th National Symposium of System Theory, Robots, Computers and Process Management*, Craiova, Romania, 1994.

[Matlab96] Cambridge Control Solutions, Newton House, Cambridge Business Park, Cowley Road, Cambridge, February 1996.

[McKerrow91] McKerrow, P.J., *Introduction to Robotics*, Addison-Wesley Publishers Ltd., 1991.

[Mead88] Mead, C.A., Mahowald, M.A., “A silicon model of early visual processing”, *Neural Networks*, 1, pp. 91-97, 1988.

[Mel95] Mel, B.W., “Planning Connectionist”, *The Handbook of Brain Theory and Neural Networks*, Arbib, M.A. (ed.), MIT Press, pp. 741-745, 1995.

[Meyer88] Meyer, W., Benedict, P., Path Planning and the Geometry of Joint Space Obstacles, *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 304-310, 1988.

[Millan92] Millán, J.R., Torras, C., “A Reinforcement Connectionist Approach to Robot Path Finding in Non-Maze-Like Environments”, *Machine Learning*, Special Issue on Reinforcement Learning, Vol. 8, No. 3/4, pp.139-171, May 1992.

[Miller90] Miller, W.T., An, E., Glanz, F.H., Carter, M.J., “The Design of CMAC Neural Networks for Control”, *Proceedings of Sixth Yale Workshop on Adaptive and Learning Systems*, Yale University, August 15-17, 1990.

[Mitchell96] Mitchell, R.J., Keating, D.A., Goodnew, I.C.B., Bishop, J.M., “Multiple Neural Network Control of Simple Mobile Robots”, *Proceedings of the Fourth IEEE Mediterranean Symposium on New Directions in Control and Automation*, Crete, pp. 271-275, June 1996.

[Morasso95] Morasso, P., Giuffrida, F., Vercelli, G., Zaccaria, R., “Safety of man-robot interaction in a domotic application”, *Proceedings of* the *5th International Conference on Artificial Neural Networks (industrial section)*, Paris, France, October 9-13, 1995.

[Morgan95] Morgan, G., Austin, J., “Safety Critical Neural Networks”, *Proceedings of the International Conference on ‘Artificial Neural Networks’*, Cambridge, UK, pp. 212-217, 26-28 June 1995.

[Musavi92] Musavi, M.T., Ahmed, W., Chan, K.H., Faris, K.B., “On the Training of Radial Basis Function Classifiers”, *Neural Networks*, Vol. 5, pp. 595-603, Pergamon Press Ltd., 1992.

[Nauck94] Nauck, D., Klawonn, F., Kruse, R., *Neuronale Netze und Fuzzy Systeme*, Vieweg, Braunschweig/Wiesbaden, Germany, 1994 (in German).

[Newman91] Newman, W.S. and Branicky, M.S., “Real-Time Configuration Space Transforms for Obstacle Avoidance”, *The International Journal of Robotics Research*, Vol. 10, No. 6, 1991.

[Nguyen90] Nguyen, D., Widrow, B., “The Truck Backer-Upper: An Example of Self-Learning in Neural Networks”, *Neural Networks for Control*, Miller, W.T., Sutton, R.S., Werbos, P.J., (eds.), The MIT Press, Cambridge, USA, 1990.

[Noble64] Noble, B., *Numerical Methods II: Differences, Integration and Differential Equations*, Oliver and Boyd Ltd., UK, 1964.

[Palm91b] Palm, R., “Fuzzy-Control: Grundlagen und Entwicklungsmethoden”, *Informationstechnik (it)*, 33, Oldenbourg Verlag, June 1991 (in German).

[Racz92] Racz, J., Mieleszczenko, W., “Mobile Robot for Inspection of Nuclear Installations”, *Archiwum Budowy Maszyn*, Vol. XXXIX, pp. 79-90, 1992.

[Ralli94] Ralli, E., Hirzinger, G., "Fast Path Planning for Robot Manipulators Using Numerical Potential Fields in the Configuration Space", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Munich, Germany, pp.1922-1929, Sept. 12-16, 1994

[Ralli96] Ralli, E., Hirzinger, G., "A Global and Resolution Complete Path Planner for up to 6dof Robot Manipulators", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Minneapolis, Minnesota, USA, pp. 3295-3302, April 23-27, 1996

[Ramo94] Ramo, S., Whinnery, J.R., Duzer, T. van, *Fields and Waves in Communication Electronics*, John Wiley & Sons, Inc., 1994.

[Redmill95] Redmill, F. (editor), “Medical safety systems”, in *Computing & Control - Engineering Journal*, IEE Publication, Vol. 6, No. 5, pg. 202-232, October 1995.

[Reignier93] Reignier, P., “Fuzzy Logic Techniques for Mobile Robot Obstacle Avoidance”, *Proceedings of the International Workshop on Intelligent Robotic Systems ’93*, Zakopane, pp. 187-197, July 1993.

[Reignier94] Reignier, P., “Molusc: An Incremental Approach of Fuzzy Learning”, *Proceedings of the International Workshop on Intelligent Robotic Systems ’93*, Grenoble, pp. 179-187, July 1994.

[Reimer86] Reimer, M., *Höhere Mathematik IV*, Lecture Script, Lehrstühle Mathematik III & VIII, University of Dortmund, 1986 (in German).

[Risse95] Risse, W., Fink, B., Hiller, M., “Multisensor-based Control of a Redundant SCARA Robot”, *Proceedings of Ninth World Congress on the Theory of Machines and Mechanisms*, Milano, Italy, 30 August - 2 September, 1995.

[Ritter92] Ritter, H., Martinetz, T., Schulten, K., *Neural Computing and Self-Organizing Maps*, Addison-Wesley Publishing Company, 1992

[Roska96] Roska, T., “Cellular Neural Networks-A Paradigm behind a Visual Microprocessor”, *Presentation at the NeuroFuzzy ’96 - IEEE European Workshop*, Prague, Czech Republic, 16-18 April, 1996.

[Roth93] Roth, H., Schilling, K., Theobald, B., “Fuzzy Control Strategies for Mobile Robots”, *Proceedings of the 9th Fachgespräch über Autonome Systems, Munich*, Germany, pp. 251-260, October 1993.

[Rucci93] Rucci, M., Dario, P., “Active Exploration in Robotic Tactile Perception”, *Proceedings of the International Workshop on Intelligent Robotic Systems*, Zakopane, pp. 212-223, 1993.

[Schmidt95] Schmidt, H., Velastin, S.A., “Fuzzy Controlled Robot Gripper System for Slipping of Objects”, *Second Latin American Seminar on Advanced Control (LASAC 95)*, Santiago, Chile, 26-29 September 1995.

[Schuster94] Schuster, C., Schmitz, T., Hiller, M., “Investigation of Stability and Robustness of a Fuzzy Traction Control System”, *presented at WW Workshop on Fuzzy Logic and Neural Networks*, Nagaya, Japan, 1994.

[Serac96] Serac, A., Roth, H., “Design of a complex rule-based controller for a biotechnological process”, *Neural Network World*, Vol.6, No. 4, pp. 701-709, 1996.

[Sharkey96] Sharkey, N.E., Heemskerk, J.N.H., Neary, J., “Training Artificial Neural Networks for Robot Control”, *Proceedings of the 2nd International Conference on Engineering Applications of Neural Networks*, pp. 190-196, London, UK, 17-19 June, 1996.

[Siemiatkowska94] Siemiatkowska, B., “A Highly Parallel Method for Mapping and Navigation of An Autonomous Mobile Robot”, *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, California, Vol.4, pp. 2796-2801, May 8-13, 1994.

[Siemiatkowska94b] Siemiatkowska, B., “Cellular neural network for mobile robot navigation”, *Proceedings of the Third IEEE International Workshop on Cellular Neural Networks and their Applications*, Rome, Italy, pp. 285-290, December 18-21, 1994.

[Skubic93] Skubic, M., Graves, S., Mollenhauer, J., "Design of a two-level fuzzy controller for a reactive miniature mobile robot", *Proceedings of the Third International Conference on Industrial Fuzzy Control and Intelligent Systems*, Houston, TX, USA, December 1993.

[Song92] Song, K., Tai, J., “Fuzzy Navigation of a Mobile Robot”, *Proceedings of the 1992 IEEE/RJS International Conference on Intelligent Robots and Systems*, Raleigh, NC, pp. 621-627, July 1992.

[Sugeno84] Sugeno, M., Murakami, K., “Fuzzy Parking Control of Model Car”, *Proceedings of 23rd IEEE Conference on Decision and Control*, Las Vegas, USA, pp. 902-903, December 1984.

[Sulzberger93] Sulzberger, S.M., Tschichold-Gürman, N.N., Vestli, S.J., “FUN: Optimization of Fuzzy Rule Based Systems Using Neural Networks”, *Proceedings of the IEEE International Conference on Neural Networks*, San Francisco, CA, UAS, 28 March - 1 April, 1993.

[Sutton90] Sutton, R.S., “Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming”, *Proceedings of the Seventh International Conference on Machine Learning*, pp. 216-224, Morgan Kaufmann, 1990.

[Sutton91] Sutton, R.S., “Planning by Incremental Dynamic Programming”, *Proceedings of the Ninth Conference on Machine Learning*, pp. 353-357, Morgan Kaufmann, 1991.

[Tarassenko91] Tarassenko, L., Blake, A., “Analogue computation of collision-free paths”, *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, Sacramento, CA, April 1991.

[Tschichold96] Tschichold-Gürman, N., “The Neural Network Model RuleNet and its Application to Mobile Robot Navigation”, Special Issue of *Methods for Data Analysis of Fuzzy Sets and Systems*, 1996 (*in press*).

[Tschichold96b] Tschichold-Gürman, N., “RuleNet: A Knowledge-Based Neural Network Model with Application Examples in Mobile Robotics”, PhD Thesis at the Swiss Federal Institute of Technology, ETH Zürich, Diss. ETH No. 11356, 1995

[Vestli94] Vestli, S.J., Tschichold-Gürman, N., Andersson, H., “Learning control and localisation of mobile robots”, *10. Fachgespräch über Autonome Mobile Systeme (AMS’94)*, Stuttgart, Germany, 13-14 October, 1994.

[Winkelmann96] Winkelmann, R., “Building Configuration Space for a Two-Link, Planar Manipulator for Robot Motion Planning”, Report for ERASMUS-Project, Department of Mechanical Engineering, King’s College London, UK, September 1996.

[Wolff68] Wolff, I., *Grundlagen und Anwendungen der Maxwellschen Theorie I & II*, Bibliographisches Institut, B.I.-Wissenschaftsverlag, Mannheim/ Wien/ Zürich, 1968 (in German).

[Zadeh96] Zadeh, L., “The Pivotal Role of Information Granulation in Fuzzy Logic, Computing with Words and Decision Analysis”, *Proceedings of EUFIT '96 - The Fourth European Congress on Intelligent Techniques and Soft Computing*, Aachen, September 2-5, 1996.

[Zavlangas96] Zavlangas, P., “Obstacle Avoidance for Robotic Manipulators Using Fuzzy Logic”, Masters’ Thesis at the Dept. of Electronic and Electrical Eng., King’s College London, 1996.

[Zicky95] Zicky, A., “The Model of the Video Safety System for Industrial Robots”, (Internal Report), Central Institute for Labour Protection, Department of Safety Engineering, Czerniakowska 16, 00-701 Warszawa, Poland, 1995.

[Zimmer94] Zimmer, U.R., Fischer, C., von Puttkammer, E., “Navigation on Topologic Feature-Maps”, *Proceedings of IIZUKA’94*, Fukuoka, Japan, 1-7 August, 1994.

[Zimmer94c] Zimmer, U.R., von Puttkammer, E., “Comparing World-Modelling Strategies for Autonomous Mobile Robots”, *Proceedings of IWK’94*, Ilmenau, Germany, September 27-30, 1994.