Development of on-line motion planning for industrial robot in dynamic environments

Wu, Xiaojun

2007

Wu, X. (2007). Development of on-line motion planning for industrial robot in dynamic environments. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/5410>

<https://doi.org/10.32657/10356/5410>

Nanyang Technological University

Downloaded on 26 Apr 2023 17:37:56 SGT

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

**DEVELOPMENT OF ON-LINE MOTION
PLANNING FOR INDUSTRIAL ROBOT IN
DYNAMIC ENVIRONMENTS**

WU XIAOJUN

**SCHOOL OF MECHANICAL AND AEROSPACE
ENGINEERING
2007**

DEVELOPMENT OF ON-LINE MOTION PLANNING FOR
INDUSTRIAL ROBOT IN DYNAMIC ENVIRONMENTS

WU XIAOJUN

2007

Development of On-line Motion planning for Industrial Robot in Dynamic Environments

Wu Xiaojun

School of Mechanical and Aerospace Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2007

Acknowledgements

First and foremost, I would like to express my sincere and grateful thanks to my supervisor, Assoc. Prof. Li Qing, for her invaluable guidance throughout my Ph.D research and her great encouragement and help in improving my English. The same sincere thanks should be given to Asst. Prof. Heng Kok Hui, my co-supervisor, for his insightful advices on the project.

Also special thanks to Asst. Prof. Jiong Tang from University of Connecticut, USA, for his many instructive comments and suggestions on my research and valuable efforts to revise the papers.

Thanks Mr. S.A. Ehmann, a former researcher in University of North Carolina at Chapel Hill, who kindly provided me the source code of SWIFT++ software, which is used in this thesis for distance calculation.

Then, I am deeply grateful to my parents for their continuous understanding and steadfast support.

My thanks are also dedicated to all technicians in Mechatronics & Control Lab, they are Ms. Lee Koon Fong, Ms. Pamela Loh Jee Luan, Mr. Lim Chai Lye and Mr. Ng Tian Seng, for their kind help and supports.

At last, thanks are due to my friends in Mechatronics & Control Lab and Robotics Research Center, NTU, for the valuable interactions which have helped to make my Ph.D research in NTU an enjoyable and fruitful endeavor.

Abstract

Many advanced robotic tasks are often completed in non-structured and dynamic environments, with events whose occurrence in time and space are not precisely known ahead of time. The environmental changes put forward new demands that robot motions must be both planned and executed in real time. However, most existing motion planners are only able to generate path plans in an offline manner. This research project thus aims to develop several practical on-line motion planners for typical industrial robots operating in unstructured dynamic working environments. In this thesis, several efficient and effective motion planners have been proposed and verified through analysis, simulation and experimental studies based upon the test platform of an educational robot — ESHED SCORBOT ER4pc with five degree-of-freedom, and an industrial robot — PA-10 with seven degree-of-freedom. In the first part of this thesis, three planner versions executed in the workspace have been developed in sequence for collision-free and optimal motion, each with improved performance. In the second part of this thesis, in order to seek complete on-line motions, the global configuration space (C-space) connectivity with respect to all of the possible obstacle positions are pre-computed in the offline stage; in the online stage, the real-time obstacles presented in the environment are accounted for, and heuristics including hierarchical C-space representation and multi-resolution searching strategy are applied to speed up the path searching. In the third part of this thesis, a compound planner design combining both the workspace planning and C-space planning is developed, in which the global collision-free path is found using C-space searching; meanwhile, planning is executed in the workspace so that the robot links are driven to avoid dynamic obstacles. Such a compound planner is verified in both efficiency and robustness for motion planning of typical industrial robots operating in dynamic environments. Lastly, experiments are executed to verify the practical feasibility of the proposed motion planners for a real robot.

Table of Content

Abstract.....	I
Table of Content	II
Chapter 1 Introduction	1
1.1 Background of Motion Planning for Industrial Robots.....	1
1.2 The Research Objective.....	4
1.3 Approaches	6
1.3.1 Planning in the Workspace	6
1.3.2 Planning in the C-space.....	10
1.3.3 Compound Motion Planning	12
1.4 Experimental Verification.....	13
1.5 Overview of Thesis	13
Chapter 2 Literature Survey	15
2.1 General Motion Planning Methods	15
2.2 Planning in Dynamic Environments	20
2.2.1 Planning Methods Incorporating Time into the Configuration Space.....	21
2.2.2 Reactive Planning.....	22
2.2.3 Sampling-Based Roadmap Methods	23
2.2.4 Other Approaches	25
2.3 Summary	26
Chapter 3 Motion Planning in Workspace.....	27
3.1 Introduction	27
3.1.1 Multi-agent Strategy for Manipulator Motion Planning	28
3.1.2 Fuzzy Reasoning	29
3.2 Planner Design	30
3.2.1 Higher-level Planner Design	31
3.2.2 Lower-level Planner Design.....	37
3.2.3 More Design Considerations.....	40
3.2.3.1 Possible Collision Types	40

3.2.3.2 Integrating a Backtracking Mechanism in Unsuccessful Trials	41
3.3 Simulation	43
3.3.1 Simulation Case Employing the SCORBOT robot: Single Static Obstacle around the Forearm	44
3.3.2 Simulation Case Employing PA-10 Robot: Single Static Obstacle around the Tool-tip	46
3.4 Conclusions and Discussion	48
Chapter 4 Improved Workspace Planning.....	50
4.1 The Second Version of the W-space Planner.....	50
4.1.1 Planner Architecture	50
4.1.2 Development of Vector-format Fuzzy Reasoning for the End-effector	51
4.1.2.1 Mathematical Description of the Vector-format Fuzzy Logic Reasoning.....	53
4.1.2.2 Applying the Vector-format Fuzzy Approach to End-effector Navigation	61
4.1.3 Reactive Approach for Links to Avoid Obstacles	68
4.1.4 General-inverse Jacobian	71
4.1.5 Simulation	71
4.1.5.1 Case 1 - Environment Containing Single Dynamic Obstacle	73
4.1.5.2 Case 2 - Environment Involving a Large Wall	75
4.1.5.3 Case 3 - Complex Scenario Involving Non-convex Obstacles	77
4.1.6 Summary of the Second Version of the W-space Planner	78
4.2 The Third Version of the W-space Planner.....	78
4.2.1 Wrist-decouple Structure and Decomposition Strategy	79
4.2.2 Planning Position Movement of the Wrist Center.....	81
4.2.2.1 Representing the W-space by An Octree.....	82
4.2.2.2 On-line Path Searching.....	85
4.2.2.3 Obstacle Avoidance of Positioning Links	92
4.2.3 Planning the Gripper Orientation	92
4.2.4 Simulation	93
4.2.4.1 Retesting the “Star” Scenario.....	94
4.2.4.2 Environment Containing Multiple Dynamic Spheres	97

4.2.4.3 Simulation with a 7-DOF Robot Prototype.....	101
4.2.5 Conclusion and Discussion of the Third Version of the W-space Planner	104
Chapter 5 Configuration-Space Construction.....	106
5.1 Introduction	106
5.2 Off-line Phase.....	109
5.3 On-line C-space Construction	112
5.4 Representation Scheme	113
5.4.1 Representations of the Robot and the Obstacle in W-space.....	113
5.4.2 C-space Representation by Non-uniform 2^m -tree.....	116
5.4.2.1 Non-uniform 2^m -tree	116
5.4.2.2 Encoding the C-obstacle Maps.....	120
5.4.2.3 Decoding and Superimposing C-obstacle Maps	122
5.4.3 Techniques to Reduce the Space Consumption of COD	123
5.4.3.1 Optimal Resolution Selection.....	123
5.4.3.2 Reference C-obstacle Map	124
5.4.3.3 Interpolation of the Reference C-obstacle Maps.....	124
5.4.3.4 Comments on Reference C-obstacle Maps and Their Interpolation	128
5.5 Simulation Results.....	128
5.5.1 C-obstacle Construction for RRR Robot.....	128
5.5.2 C-space Construction for a 5-DOF Robot.....	129
5.6 Complexity Analysis	133
5.7 Conclusion.....	133
Chapter 6 Path Searching in Configuration Space	135
6.1 Neighbour Searching in a Non-Uniform 2^m -tree.....	135
6.2 On-line Path Searching under Multi-resolution Strategy	138
6.2.1 Maintaining the OPEN List by Binary Heaps.....	142
6.2.2 Lazy Evaluation of Neighbours.....	143
6.2.3 Self-collision	144
6.2.4 Path Generation and Smoothing.....	144
6.3 Simulation	145

6.3.1 Case 1 - Simulation in Static Environment	145
6.3.2 Case 2 - Dynamic Wall.....	147
6.3.3 Case 3 – Static and Dynamic Obstacles	149
6.3.4 Case 4 - Simulation with Two Robots.....	150
6.4. Conclusion.....	152
Chapter 7 Compound Motion Planner.....	154
7.1 Compound On-line Planning Design	154
7.2 Simulation	155
7.2.1 Simulation Case with Multi Spheres.....	155
7.2.2 Simulation Case with Two Robots	159
7.2.3 Simulation Case with Static Walls and Multi Dynamic Cylinders.....	162
7.2.4 Simulation with 7-DOF PA-10 Robot	165
Chapter 8 Experimental Verification	172
8.1 Introduction	172
8.2 Testing Hardware	173
8.2.1 New Interface for Robot Control	175
8.2.2 Moving Obstacle Sub-system.....	177
8.3 Control Algorithm and Software Implementation.....	179
8.3.1 Control Scheme	179
8.3.2 Software Development.....	179
8.4 Testing Results	180
8.4.1 Comparison of the Calculated and the Real Robot Motion	180
8.4.2 Motor Trajectories.....	185
8.4.3 Comments of the Testing Result	189
Chapter 9 Conclusion.....	190
9.1 Work Accomplished in this Research.....	190
9.1.1 Planning in the Workspace	190
9.1.2 Planning in the Configuration Space.....	192
9.1.3 Compound Planner Design.....	193
9.1.4 Experimental Verification.....	194

9.2 Contributions	194
9.3 Future Work.....	196
Appendix	199
Appendix A - Specification of the SCORBOT ER4pc Robot.....	199
Appendix B - Specification of PA-10 Robot.....	203
Appendix C - Calculating General-inverse Jacobian	206
Appendix D - Basic A* searching algorithm	213
Appendix E – Class definition of motion planning and motion control	214
References	220
Publications.....	233

List of Figures

Figure 1.1 Parts transfer	4
Figure 3.1 Structure of fuzzy logic.....	29
Figure 3.2 Structure of the designed motion planner with two levels.....	31
Figure 3.3 Illustration of the behavior collection.....	32
Figure 3.4 Membership functions of the higher-level planner.....	34
Figure 3.5 Membership functions of the lower-level planner.....	38
Figure 3.6 Possible collision types.....	40
Figure 3.7 Initial and goal configuration of static simulation case 1	45
Figure 3.8 Motion planning process of the static simulation case 1	45
Figure 3.9 Joint trajectory of static simulation case 1	46
Figure 3.10 Motion planning process of the static simulation case 2	47
Figure 3.11 Joint trajectory of static simulation case 2.....	48
Figure 4.1 Graphical illustration of the orientation membership function “UP” ($\sigma = 0.3$).....	56
Figure 4.2 Graphical representation of the fuzzy inference for rule 1 ($\sigma = 0.5$)	57
Figure 4.3 Graphical representation of the fuzzy inference for rule 2 ($\sigma = 0.5$)	58
Figure 4.4 The union of the two rules	58
Figure 4.5 Illustration of the spherical coordinate ($ \bar{P}_i = 1$).....	60
Figure 4.6 The output vector after defuzzification.....	61
Figure 4.7 The two-level fuzzy planner in vector-format for navigation of the end-effector ..	61
Figure 4.8 The direction for the end-effector to move round an obstacle.....	63
Figure 4.9 The effect of the repulsive evading of a link away from an obstacle	69
Figure 4.10 Membership function of input and output linguistic variables.....	70
Figure 4.11 Structure of the second version of W-space planner.....	72
Figure 4.12 Motion sequence of simulation case containing a single dynamic obstacle.....	74
Figure 4.13 Joint trajectory VS time of the 1 st simulation case	75
Figure 4.14 Simulation scenario with a large scale wall.....	76
Figure 4.15 Joint trajectory VS. time of the 2 nd simulation case.....	77

Figure 4.16 Simulation scenario experiencing navigator failure guiding end-effector out of non-convex start position	78
Figure 4.17 The wrist-decompose structure of typical industrial robots.....	80
Figure 4.18 The octree shape representation with bit encoding.....	83
Figure 4.19 The complete representation of the octree in Figure 4.18(d).....	84
Figure 4.20 Searching free neighbours in the octree sample	88
Figure 4.21 The structure of the third version of the W-space planner.....	95
Figure 4.22 Pre-processing the obstacle surfaces to obtain control vertexes.....	95
Figure 4.23 Motion sequence of the static simulation case.....	96
Figure 4.24 Joint trajectory of static simulation case.....	97
Figure 4.25 Screenshots of the dynamic simulation case with multi spheres.....	99
Figure 4.26 Time performance of the simulation case 2	101
Figure 4.27 Dynamic simulation case	102
Figure 4.28 Simulation result of case 3	104
Figure 4.29 Collision-free path for the end-effector is not feasible for the robot	105
Figure 5.1 Non-existence of general heuristics for C-space construction.....	108
Figure 5.2 Illustration of the C-obstacle map concept using a planar RR manipulator	111
Figure 5.3 C-obstacle corresponding to different representation mode in W-space	115
Figure 5.4 The uniform and reasonable non-uniform discretization (from [127]).....	117
Figure 5.5 Adjacent link positions with swept area zero between them (from[128])	118
Figure 5.6 Quad tree demo for non-uniform 2^m -tree	119
Figure 5.7 Non-uniform space decomposition under another convention	120
Figure 5.8 Representing the quadtree in Figure 5.6(c) by scheme c	121
Figure 5.9 The complete 2^m -tree after bit decoding	123
Figure 5.10 Calculated C-obstacle under different resolutions	125
Figure 5.11 C-space calculated based on neighbour encoding	126
Figure 5.12 Demonstration of C-obstacle interpolation in a 2D case	127
Figure 5.13 Another interpolation approach	127
Figure 5.14 Simulation results of a spatial RRR manipulator.....	129
Figure 5.15 Decomposition result of the SCORBOT robot at one configuration during off-line	

COD creation.....	130
Figure 5.16 ESHED SCORBOT-ER4pc Robot and the obstacle.....	131
Figure 5.17 The geometric decomposition result of obstacles.....	132
Figure 6.1 Sample space demonstrating the directions of free neighbours.....	136
Figure 6.2 A non-uniform quadtree sample demonstrating the free-neighbour searching.....	137
Figure 6.3 Flowchart of multi-resolution path planning in the C-space	141
Figure 6.4 Sample case verifying the A* algorithm under multi-resolution strategy	142
Figure 6.5 Static simulation scenario	146
Figure 6.6 Dynamic simulation scenario- Screenshot of simulation case 2.....	148
Figure 6.7 Result of the simulation case 2	149
Figure 6.8 Collision occurred between the robot arm and spheres	151
Figure 6.9 Demonstration of the non-sensitivity of the developed C-space planner	151
Figure 6.10 Simulation case including two robots.....	152
Figure 7.1 The structure of the compound motion planner.....	156
Figure 7.2 Screenshots in the simulation case including multi moving spheres.....	157
Figure 7.3 Planning results of the simulation case with multi moving spheres.....	159
Figure 7.4 Screenshot in the simulation case including two robots.....	160
Figure 7.5 Time performance of the simulation case with two robots.....	162
Figure 7.6 Screenshot in simulation case including static walls and dynamic cylinders.....	163
Figure 7.7 Planning Results of the simulation case including static walls and dynamic cylinders	165
Figure 7.8 Flowchart of the compound planner for PA-10 Robot.....	167
Figure 7.9 Screenshots in simulation case 4 (with obstacle avoidance)	168
Figure 7.10 Planning results of the simulation case 4.....	169
Figure 8.1 Block diagram of a typical testing system for robot motion planner.....	172
Figure 8.2 Proposed motion control system.....	174
Figure 8.3 Layout of the actual testing rig	174
Figure 8.4 Humusoft MF 614 multifunction I/O card.....	176
Figure 8.5 Four A-M-C 12A8 servo amplifiers driving the servo motors	177
Figure 8.6 The moving obstacle.....	177

Figure 8.7 The stepper driver	178
Figure 8.8 Synchronized real and visualized robot motion.....	181
Figure 8.9 Comparison of the calculated motion sequences and the real robot motion	184
Figure 8.10 Joint trajectory and joint error of the 1 st axis	185
Figure 8.11 Joint trajectory and joint error of the 2 nd axis	186
Figure 8.12 Joint trajectory and joint error of the 3 rd axis.....	187
Figure 8.13 Joint trajectory and joint error of the 4 th axis.....	188
Figure A.1 Appearance of the SCORBOT Robot arm and its control box	199
Figure A.2 Link coordinates of ESHED SCORBOT ER4pc Robot	201
Figure B.1 Mechanical Configuration of the PA-10 Manipulator.....	203
Figure B.2 Link Coordinates of PA-10	205
Figure C.1 Illustration of link frames and reference point.....	206

List of Tables

Table 3.1 Motion variables.....	33
Table 3.2 Fuzzy variables of the higher-level planner	34
Table 3.3 Fuzzy rules of the higher-level planner	36
Table 3.4 Fuzzy variables of the lower-level planner	37
Table 3.5 Fuzzy rules of <i>goal-approaching</i> behavior	39
Table 3.6 Fuzzy rules of <i>slide</i> behavior.....	39
Table 3.7 Fuzzy rules of <i>obstacle_evade</i> behavior.....	40
Table 4.1 Membership functions for linguistic variable - orientation.....	55
Table 4.2 Fuzzy rule base	65
Table 4.3 Fuzzy rules.....	70
Table 4.4 Storage requirement of three octree implementations.....	84
Table 4.5 Procedures to find the directions for free neighbour searching.....	87
Table 5.1 Process to determine the resolution leading to reasonable W-space coverage.....	119
Table 5.2 Comparison of three representation schemes	122
Table 5.3 Settings and calculation result of the 3-DOF robot.....	129
Table 5.4 Settings and calculation result employing the SCORBOT Robot.....	132
Table 6.1 Procedures to find all directions for free neighbour searching.....	136
Table 7.1 Parameter settings to calculate the <i>COD</i> for the PA-10 robot.....	166
Table 7.2 Comparison of the developed motion planners	171
Table 8.1 Component list for experimental test	175
Table 8.2 Parameters of the PD controllers.....	179
Table A.1 Mechanical specifications of ESHED SCORBOT ER4pc Robot	200
Table A.2 D-H parameters of SCORBOT Robot	201
Table B.1 Mechanical Specifications of PA-10 Robot.....	203
Table B.2 Kinematics Parameters (D-H Representation) of PA-10 Robot	205

Chapter 1 Introduction

1.1 Background of Motion Planning for Industrial Robots

The application of industrial robots has achieved the aims of increasing production whilst reducing costs over many years that it has been in existence. It can also relieve human operators from tedious jobs and enable them to control at a supervisory level. In turn, this increases efficiency and repeatability by eliminating human errors [1]. Some typical industrial robotic applications are given as follows [2]:

- Robotic assembly
- Packaging
- Foundry application
- Transfer of parts and components
- Robotic material handling and machine tending
- Pick and place on the fly
- Flexible palletizing
- Robotic painting and coating
- Spot welding and arc welding

The key characteristic of industrial robots is their versatility — they can be applied to a large variety of tasks without major re-design of the mechanical structure. This versatility can be attributed to the flexible physical structure, sensing capability and programmability of robots [3].

Motion planning is one of the key requirements to realize the required versatility for creating autonomous robots. It plays a crucial role for the potential successful applications of industrial robots. For industrial robots, motion planning concerns the problem of finding the path of the robot end-effector which always has some form of constraints generated from working tasks and constraints from the robot structure too. Automatic path planning can greatly simplify the task of programming industrial robots, realize flexible production with less setup time and lower cost in the manufacturing. Motion planning enables a robot to shift from one

manufacturing task to another much faster than before. Hence, in the last twenty years, motion planning received much attention in robotic applications and researches.

Most industrial robots are used in well structured environments for simple manipulation tasks. Motion planning in this situation is also called the basic motion planning problem (BMPP) which can be defined as: given an initial configuration and a desired final configuration of the robot, find a path, starting at the initial configuration and terminating at the final configuration, while avoiding collisions with the obstacles [4]. Some of the most important motion planning achievements for BMPP and its extensions are listed as follows:

- Cell-decomposition methods

Following the concept of configuration space (C-space) proposed by Lozano Perez [5], the free space of a robot is decomposed into simple regions, e.g. in format of rectangle shape, called cells. The cells are labeled as free, obstructed, or mixed. These methods generally proceed by recursively subdividing mixed cells until a pre-determined minimal cell size is reached. After such a C-space approximation is constructed, the search will be executed in the C-space to find a path between any two cells [6-8].

- Potential field methods

Krogh [9] and Khatib [10] proposed the concept of artificial potential field for robot motion planning. In this method, the goal generates an attractive potential which pulls the robot toward it and obstacles produce a repulsive potential which pushes the robot away from them. The negative gradient of the total potential field is considered as the most promising direction of the motion [10-15].

- Sampling-based Approaches

First developed by Kavraki etc.[16], the main idea of sampling-based motion planning is first to form a graph for the robot working environment, termed roadmap, by sampling the configuration space. In this roadmap, the nodes correspond to collision-free configuration and the edges represent path availability between node pairs. Once a roadmap is constructed, a

randomized searching or probabilistic searching approach is used to find the collision-free way from the roadmap [4, 16-22].

However, many advanced robotic tasks, like robot guidance, tele-operation, assembly and disassembly, palletizing/packaging objects, medical surgery and so on, are often completed in non-structured or dynamic environments. These applications involve events whose occurrence in time and space are not precisely known ahead of time. Among these tasks, some involve *quasi-dynamic* changes, which mean that the working environment does not change within one operation cycle, but it varies from one cycle to another. Some examples are given below.

- In shipyards, work-pieces with variations in the shape and size of the webs, stiffeners and plates need to be transferred and welded using robots. In this application example, the manipulation and welding tasks vary after each robot execution cycle, so the kinematic chain of the robot, including the gripped object, changes in different operation cycles. However, the environment keeps static within one single cycle [23].
- Figure 1.1 shows a robot application example in the food industry [24]. During the working process, parts are transferred into the container from the outside or inversely moved out of the container using a robot. For picking and placing operations in this task, the initial and goal configurations of the robot are different and the relative positions between robot links and the parts to be picked up in the container also vary. However, similar as the previous example, the environment does not change within one single operation cycle.

Apart from these situations, there exist applications where the environment keeps changing. In these *dynamic* environments, changes are mainly caused by dynamic obstacles with unknown or partially unknown movements. Typical examples include a pick-and-place task on a conveyor belt where moving objects result in a changing environment, and the multi-robot cooperation where each manipulator becomes a dynamic obstacle for its partners.

Figure 1.1 Parts transfer

The dynamic changes in the environment put forward new demands to motion planning. In order to carry out the manipulation efficiently, timing is critical, which means motions must be both planned and executed fast enough in response to the changes. However, most existing motion planners are only able to generate path plans in an offline manner [25].

Among these planners, the time and space complexity of the methods based on cell decomposition grows quickly when the dimension m of the configuration space increases. In practice, these methods are realistically applicable only when the dimension is small enough (usually $m \leq 4$) [7]. For the potential-field approaches, typically they depend only on the contents in the neighbourhood of the current configuration of the robot, thus they suffer from local minima, which may cause the robot to be trapped in configurations other than the goal. On the other hand, the sampling-based method makes motion planning less dependant on the number of degree-of-freedom (DOF); however, a relative long time is required for the roadmap pre-computation phase [26]. Typically several seconds are needed for typical industrial robotic applications [16]. Furthermore, the sampling-based motion planning approaches often assume that the environment is static, which does not agree to a changing environment [27]. All these facts determine that the sampling-based methods are not directly applicable to real-time applications [28].

1.2 The Research Objective

From the above discussions, it is clear that the challenging issue in motion planning lies in

increasing its efficiency for real-time applications. It is important to note that typical industrial robotic applications involving a changing environment have some particular features: the robots have about six DOF since a robot with six DOF is able to arbitrarily position and orient an object within the robot operational space [29]; most of these robots are designed with a “wrist-decouple” structure so that the robot position movement can be separated away from the orientation part; more often, there is a relative wide clearance between the robot and obstacles; and the environmental changes are relatively slow with respect to the robot speed. The objective of this research is thus to develop a practical on-line motion planner for typical industrial robots in unstructured dynamic working environments. In order to achieve real-time motion, the features of typical industrial robotic applications as described above will be considered during the design phase.

It is also worth noting that motion planning in static environments can be guaranteed to find a solution if one exists at time t_0 , whereas motion planning in dynamic environments is essentially intractable [30, 31], i.e., the solution at t_0 may not exist at a later time because of the evolution of the environment. Taking into account the time-variant environment in which the robot operates, the motion planner should be able to plan the path in real time. Practically, it needs to update the path at small enough intervals. This is the main feature of the motion planning in dynamic environments. Then the planning action at time $t_i = t_{i-1} + T$, where T refers to the interval between two consecutive planning iterations, can be denoted as

$$\begin{cases} \Delta q_{t_i} = \text{Plan}(A(q_{t_i}), B(t_i)) \\ q_{t_i} = q_{t_{i-1}} + \Delta q_{t_i} \end{cases} \quad (1.1)$$

where Δq_{t_i} is the joint step change for the current time moment, $A(q_{t_i})$ is subset of the workspace, W , occupied by robot A when at configuration q_{t_i} , and $B(t_i)$ is another subset of the workspace occupied by obstacles. The planning action expressed in Equation 1.1 is referred to a *planning cycle* in this thesis. Accordingly, a qualified result of such a motion planning problem is a sequence of time-dependent collision-free configurations, instead of a pure geometric path, which can be formulated as

$$MP(A(q_i), B(t)) = \{q(t) \mid (q(t_0) = q_i, q(t_{end}) = q_g; \forall t_i, t_0 < t_i < t_{end}, B(t_i) \cap A(q_i) = \emptyset\} \quad (1.2)$$

Here $A(q_i) \cap B(t_i) = \emptyset$ denotes configuration q_i is collision-free. The ultimate motion sequence $q(t)$ can also be expressed using a single variable representing the distance traveled along the trajectory. We denote this variable by λ . It ranges from 0 to 1, where $\lambda = 0$ and $\lambda = 1$ correspond to the configurations on the source and destination vertex of the edge respectively.

With the ability to deal with both *quasi-dynamic* and *dynamic* environmental changes, the on-line motion planner is expected to make robot operations more flexible, and new tasks could be dynamically assigned to robots without interrupting current operations, hence the development time and implementation cost can be reduced significantly.

1.3 Approaches

1.3.1 Planning in the Workspace

In the first stage of this research, a workspace (abbreviated as W-space and equivalent to the operational space) motion planner based on the multi-agent concept and fuzzy reasoning is proposed. The inputs to the W-space planner are variables defined in the W-space such as geometric descriptions of the robot and the obstacles, positional relationships between the robot and the obstacles, etc. Based on these input variables and the kinematic structure of the robot, the W-space planner is able to generate a collision-free and optimal motion for the robot.

One of the most attractive merits for motion planning in the W-space is its low computational complexity which arises from the simplicity in positional relationship description. The positional relationships between the robot and the obstacles are directly obtainable, such as from a sensor system, thus planning in the W-space is a straightforward task, and neither pre-computation nor additional transformation is needed. The low computational complexity makes the W-space approach a promising framework for motion planning in dynamic

environments.

A multi-agent system is defined as a system which is composed of more than two agents and performs a given task by cooperation among the agents. Multi-agent robotic systems have the potential to accomplish some tasks more efficiently than a single robot system, such as in robot soccer [32]. Based on this concept, in the developed W-space planner, each joint of the robot, together with its link, is considered as an agent. Each agent is assigned different roles according to the situation at a particular operation moment.

Fuzzy logic provides a formal methodology for representing and implementing the human experts' heuristic knowledge and perception-based actions. The main advantage of a fuzzy strategy lies in the ability to extract heuristic rules from human experience, and to avoid the need for an analytic model of the process [33]. Moreover, the linguistic description in fuzzy logic will loosen the required accuracy of the environmental measure, thus it makes the representation in the W-space much easier [34]. Hence, a fuzzy logic reasoning will be applied in the development of the W-space planner.

In the first version of the developed W-space planner, a two-level hierarchical structure is employed. The higher level in the hierarchical structure is designed to dynamically assign each link an appropriate behavior and the lower level is designed to determine the joint speed according to the behavior assigned by the higher level. A back-tracking mechanism is integrated in order to avoid earlier unsuccessful trials. Simulation studies based on an educational robot ESHED SCORBOT ER4pc with five DOF and an industrial robot PA-10 with seven DOF in operational scenarios with a single static obstacle are executed. The results have verified the feasibility of this planner with the advantage of low computational cost.

However, several limitations are observed in the simulation studies of this motion planner. Firstly, the planner tries to drive the robot links to approach the goal configuration in a trial-and-error manner, which does not guarantee a successful path finding. In other words, the planner is not able to predict the existence of a collision-free path in a given operation

scenario. Secondly, every time the planner tracks back to a previous configuration, if the environment then changes, the previous trials are no more meaningful, which restricts this planner version to static working environments only.

Thus the W-space planner is further improved to solve the above mentioned problems. In the second development stage, a different planner architecture is introduced, in which the agent denoting the end-effector leads the whole robot arm to move towards the goal, while the left robot links take care of obstacle avoidance. In this scheme, the position of the end-effector tip (EE tip) relative to another object or a coordinate system in the W-space is a three dimensional vector containing both distance and pointing information. In order to represent this vector efficiently and also to implement the fuzzy reasoning with a small sized rule-base, a vector-format fuzzy reasoning approach is proposed to plan the motion for the EE tip. In this new vector-format fuzzy reasoning approach, a set of new membership functions to manipulate the fuzzy vector variables is defined, and a series of new vector-format fuzzification, fuzzy inference and defuzzification approaches is also developed. For the robot links other than the end-effector, a reactive method is applied for them to quickly respond to obstacles. Employing this reactive method, a robot link which is close to some obstacles will move away from the latter with a velocity roughly in inverse proportion to the distance between them.

Simulation studies verify that, in a scenario with single moving obstacle, the second version of W-space planner succeeds in driving the robot to the destination, meanwhile keeping the robot safely away from the obstacle. Furthermore, this planner is verified to possess the ability to go around a large static wall, which is a typical situation of local minimum. However, it is also observed in simulation studies that this version of W-space planner fails to guide the robot end-effector to escape from a non-convex situation.

The first and second versions of W-space planners belong to the category of local planning, which are only suitable for simple scenarios. For more complex environments like the above non-convex situation, the global searching idea must be applied. In the third stage of the

W-space planner development, based upon the wrist-decouple structure of typical industrial robots, the motion planning problem of such type of robots is decomposed into two sub-problems: planning the positional movement of the wrist centre and adjusting the gripper orientation, both with lower degree-of-freedom than the overall one. In the first sub-problem, the global-searching based A* algorithm, with many optimality properties [35], is applied to guide the wrist center in the W-space. Here the W-space is represented and continuously updated by a hierarchical octree data structure, within which an optimal path connecting the current position of the wrist center to its destination is continuously searched by applying the A* algorithm. In the second sub-problem, following the trajectory found in the previous sub-problem, the pose of the robot gripper is locally adjusted to approach the final configuration. Furthermore, the reactive method as applied in the second planner version is also used in both sub-problems for local obstacle avoidance.

This third approach is more robust than the vector-based fuzzy motion planner developed in the second stage. Simulation studies are also executed to verify this new planner version. In the first case, the planner succeeds to lead the robot to move in the same non-convex environment as described in the previous planner version. In the second and third cases, this planner succeeds in navigating the 5-DOF SCORBOT robot and 7-DOF PA-10 robot respectively in a scenario with both a static wall and multiple moving obstacles. The time performance of this planner version is also prominent as a path connecting the current position of the wrist center to its goal position can be searched out almost instantly in each planning cycle. However, it is observed that even if a global collision-free path does exist for the wrist center, the robot links are not guaranteed as collision-free when the robot follows this path. This also reflects the fact that although the proposed planner achieves high computational efficiency, it does not possess the property of “completeness”. Therefore, in the next development stage we turn to the configuration space (C-space) to seek complete solutions.

1.3.2 Planning in the C-space

The notion of Configuration space (C-space) is defined in terms of parameters that specify the position and posture of a robot where each dimension represents a joint coordinate [5]. By this notion, the obstacles in the workspace are mapped as forbidden regions in the C-space (referred to as C-obstacle), and the complement of the C-obstacle is thus the collision-free space C_{free} . Path planning for a robot with n -DOF can thus be converted to the problem of planning a path for a point in a C-space of n dimensions. Compared with the W-space planning, the C-space motion planning is *complete*, which means a path can be found if it does exist. Especially, the cell-decomposition type of C-space planning methods are *resolution complete* [7], that is, a path can be surely found if it does exist under a pre-defined resolution.

To meet the demand for on-line motion planning, two important issues need to be addressed in the development of a C-space planner. Firstly, the connectivity in the robot C-space needs to be constructed in an on-line manner to suit the environmental changes. However, due to the exponential complexity, there is no existing on-line C-space construction method in literature after an extensive search. Thus, this forms one challenge of this research. Secondly, collision-free paths need to be continuously searched from the updated C-space as the environment changes, so the challenge to implement an efficient searching algorithm in the high dimensional C-space will greatly determine the on-line performance of the C-space planner.

In this research, we propose a strategy of pre-computing the global configuration space (C-space) connectivity with respect to all possible obstacle positions in the workspace. The proposed motion planner consists of an off-line stage and an on-line stage. In the off-line stage, the C-space connectivity information with respect to all possible obstacle positions is computed, and stored using a new hierarchical data structure named non-uniform 2^m -trees. All non-uniform 2^m -trees are organized into a database and indexed by the cells of the W-space, called C-obstacle Database (*COD*). In the on-line stage, the real obstacle cells in the

workspace are identified and the corresponding 2^m -trees from the pre-computed data source are superposed to construct the real-time C-space. The collision-free path is then searched in this hierarchical C-space by using the A* algorithm. The path searching is executed under a multi-resolution strategy, i.e., the searching starts from the lowest resolution and moves to the higher resolution only when a collision-free path cannot be found. This searching strategy can greatly reduce the on-line computational time.

The pre-computation of the *COD* in the off-line phase, clearly, does not require any prior knowledge of the obstacles in the real workspace. The time-consuming step of establishing the *COD* is realized in the off-line phase, and only the identification of maps in the *COD* is needed in the on-line phase. This method for C-space construction is general and applicable to manipulators of any kinematics structure and geometric shape. For a given robot, the corresponding *COD* is only required to be created once and can later be used in different environments. All these factors make the proposed C-space planner especially suitable for industrial robotic applications in dynamic environments.

It is worth mentioning that the proposed planner uses the non-uniform 2^m -tree data structure that yields the resolution completeness. Also, the strategy of reducing the on-line computational time by pre-computing the C-space connectivity is actually analogous to the dynamic roadmaps reported in [36, 37], whereas the difference lies in that in this research, the robot C-space is represented in a discretized format while in previous studies the C-space is represented using a sampling-based approach. On one hand, the C-space representation scheme adopted in this research could be constrained by the exponentially increasing complexity with respect to the robot DOF [7]. On the other hand, this representation scheme leads to the feasibility of heuristics like hierarchical tree representation as well as multi-resolution searching strategy, which have shown effectiveness in reducing both the required storage space and the planning complexity [38-42]. Practical industrial applications involve mostly medium and small sized robots with about six DOF. For such robots, by applying the abovementioned heuristics, the discretized C-space representation is feasible even with the exponential increase of complexity with respect to the dimension, which is

demonstrated in this thesis.

Simulations on the ESHED SCORBOT Robot in a static scenario with a narrow passage prove the global localization ability of the developed C-space planner. In another application scenario containing a moving wall (representing a dynamic obstacle), under the multi-resolution strategy, the path can be found early at a coarse resolution within a short period. Compared with the developed W-space planners, the proposed C-space planning method can thoroughly avoid local minimum. However, it is further observed respectively in a simulation case containing several moving obstacles and another simulation case with two co-operating robots that the developed C-space planner is insensitive to obstacles approaching the robot, which causes unexpected collisions. To eliminate the occurrence of such collisions, some logic that provides pre-alarming is needed to enable the planner to avoid dynamic obstacles more effectively.

1.3.3 Compound Motion Planning

The developed C-space planner is resolution complete, but it still needs an effective pre-alarming logic to improve its reactivity to dynamic obstacles; whereas the developed W-space planner is not complete, it is computationally efficient and highly reactive to the changes of obstacles. Therefore, these two planners are complementary to each other. If they can be combined appropriately, it promises to realize an on-line motion planner which is both computationally efficient and complete.

Following this idea, a compound motion planning approach is proposed in the third stage of motion planner design. With this new compound planner, the global collision-free path is searched using the developed C-space approach, and the rapid response to moving obstacles is provided using the reactive method executed in the W-space. The two simulation scenarios as studied in the pure C-space planner design are retested to verify the compound planner. The simulation results show that when the robot moves along the on-line collision-free path, it can react fast enough to avoid the other moving obstacles or the reference robot, which clearly

verifies that the compound planner has overcome the limitations observed in the pure C-space planner. Lastly, the compound planner is tested in a complex dynamic scenario similar to the work in [27], employing the SCORBOT ER4pc robot and the PA-10 robot as prototype respectively. Once again, good time performance is observed from the results.

1.4 Experimental Verification

To verify the practical feasibility of the proposed motion planners for a real robot, experiments are conducted instead of just software simulations. The SCORBOT ER4pc educational robot is chosen as the testing platform mainly due to its ready availability in the research lab. This robot has neither a control mechanism open for user adjustment nor suitable sensors for obstacle detection. To implement the computed solution, the original control box of the robot was bypassed and a new interface between the host computer and the driving motors was implemented and setup; furthermore, a simple dynamic environment was created by mounting a moving plate which was driven by a stepper motor.

In order to drive the robot to track the desired motion commands generated by the motion planner, simple PD type of controllers are designed for each joint axis. These controllers are also implemented in C++ program language.

During the experiment, the time-variant motion commands and the real robot motion were recorded. The comparison between them shows a good consistency, which demonstrates the practical feasibility and efficiency of the proposed planners.

1.5 Overview of Thesis

The thesis is organized as follows. The motivation for this research was just presented in Chapter 1, and the literature will be reviewed in Chapter 2. Chapter 3 documents the first version of the W-space planner, followed by the description of the improved W-space planners, i.e., the second and third versions of the design is described in Chapter 4. Chapter 5 presents the on-line construction of C-space and in Chapter 6, on-line path searching in the

C-space is discussed. Chapter 7 describes the development of the compound planner. In Chapter 8, experimental verification of the proposed planners is presented. Lastly, the conclusion and future works are discussed in Chapter 9.

Chapter 2 Literature Survey

This research focuses on motion planning of common industrial robots in dynamic environments. The literature survey is presented in two sections. Firstly, the main categories of existing motion planning methods are reviewed; then, motion planning in dynamic environments is examined.

2.1 General Motion Planning Methods

Motion planning, in a broad sense, refers to the ability of a robot to plan its own motions. The basic motion planning problem (BMPP) is to find a collision-free path for a robot — a rigid or articulated object — among rigid static obstacles [7, 25]. To fully realize automation of robot manipulation in a complicated working environment, motion planning is one of the core elements in robot development. Since it plays a key role for the potential successful applications of robot manipulators in industry, a number of motion planners have been proposed in the past thirty years.

Several common aspects are applied to classify the existing motion planning methods, including the completeness, the scope of information utilization, the space representation scheme, the type of environment and the time mode of planning. Each aspect will be discussed in the following;

Completeness

Completeness refers to the certainty of finding a path in a robotic application if that obstacle-free path does exist. With respect to completeness, the existing motion planners can be categorized into two groups [4].

- Exact algorithms: These algorithms can either find a solution or prove that there is no solution. Exact algorithms can be used to determine the complexity of motion planning problems. As the number of robot DOF increases, complete solutions may become computationally intractable; therefore, some weaker forms of completeness have been sought.

One such form is *resolution completeness*, which denotes that if a solution exists at a given resolution of discretization, the planner will find it. Another widely referenced form is *probabilistic completeness*, which means that the probability of finding a solution (if one exists) converges to 1 as time goes to infinity [6].

- Heuristic algorithms: On the contrary, many heuristic algorithms have been implemented to accelerate the planning process by exploring some specific properties of the planning problem. However, they may either fail to find a solution for a difficult problem, or are only able to find a poor solution. Heuristic algorithms are important in engineering applications as they provide a more practical solution at low computational cost.

Scope of Information Utilization

Based on the *scope* of information utilization, existing motion planning methods are broadly classified into the following two types.

- Global methods: This type of methods considers all the information in the environment and generally a complete model of the environment is assumed to be available. Global approaches own such an advantage that a complete trajectory from the start to the target can be computed offline. However, since much environmental information is taken into consideration during the motion planning process, usually the computational cost of global motion planning methods is high; hence, they are not suitable for fast on-line obstacle avoidance [7].
- Local methods: This type of methods, on the other hand, uses only information in the vicinity of the robot. The obvious advantage of local techniques over global ones lies in their low computational requirements. However, since only a small fraction of the environmental information is used to generate a path, the overall optimal solution may be missed and robots controlled by this type of method often fail to find trajectories between closely spaced obstacles.

Among existing local approaches, the potential field method is a typical example [10-15]. In this method, the robot is represented as a point in configuration space, moving under the influence of artificial potential forces. Typically the goal configuration generates an attractive

potential which pulls the robot toward the goal, and the obstacles in the configuration space (C-obstacle) produce a repulsive potential which pushes the robot away from them. The negative gradient of the total potential field is considered as the most promising direction of the motion. Normally, the potential field method does not include an initial processing step aiming at capturing the connectivity of the free space. Instead, at each step, they move from one configuration in the grid to another, basing their choice on the computed potential gradient. It is typical for this gradient approach to depend only on the contents of the configuration space in the neighbourhood of the current configuration of the robot. While this approach is computationally efficient, it suffers from local minima, which may cause the robot to be trapped in configurations other than the goal. Some researchers proposed harmonic potentials [11, 12] and navigation functions [14] to overcome this problem. Harmonic potentials attain their extreme values on the boundary of their domain, and obstacles represented by harmonic potentials do not induce local minima. The navigation function consists of asymptotically decaying functions that are tuned to avoid local minima. However, both functions are computationally intensive, especially for robots with many DOF in on-line planning.

Space Representation

Based on the scheme to represent the space in which the robot operates, motion planners are classified into two categories.

- **Cell-decomposition Representation**

In this representation, the free space of a robot is decomposed into simple regions called cells, and a non-directed graph representing the adjacent relation among the cells, called connectivity graph, is built such that a path between any two configurations can be generated. Cell decomposition distinguish themselves from other methods in that they can be used to achieve coverage, i.e., to find an exhaustive walk through the adjacency graph [6]. Cell decomposition methods can be further broken down into exact and approximate methods:

- Exact cell decomposition: The free space (C_{free}) of a robot is decomposed into a collection of non-overlapping regions, called cells, whose union is exactly C_{free} or its closure. The exact cell decomposition method possesses the completeness in path searching.
- Approximate cell decomposition: The cells are required to have a simple pre-specified shape, e.g. a rectangle shape and their union is strictly included in the free space, thus a conservative approximation of this space is constructed [7, 8]. The cells are labeled as free, obstructed, or mixed. These methods generally proceed by recursively subdividing mixed cells until a pre-determined minimal cell size is reached. Approximate cell decomposition methods are only complete to the resolution of the smallest allowed cell.

In practice, the time and space complexity of the cell-decomposition methods grows drastically as the dimension m of the configuration space increases. These methods are realistically applicable only when this dimension is small enough (usually $m \leq 4$). Specific modifications exploiting the structure of a particular task domain may possibly be used to develop planners working in higher-dimensional spaces. For example, some methods operate in a hierarchical fashion, by generating an initial coarse decomposition and then locally refining this decomposition until a free path is found or the decomposition becomes too fine. Exact methods are mathematically more complicated than approximate ones. Hence the latter are usually easier to implement. Popular cell decomposition methods include the *trapezoidal decomposition* [43] which relies heavily on the polygonal representation of the planar configuration space, and *Morse Decomposition* [44] which allows for representations of non-polygonal and non-planar spaces.

▪ Sampling-based Approaches

The approaches in this category try to capture the connectivity of C_{free} by sampling the configuration space and connecting the samples in free space by local paths, thus creating a network of curves, namely, a roadmap. Samples and local paths are checked for collision using a fast collision checker which avoids the prohibitive computation of an explicit

representation of the free space. Once a roadmap R is constructed, path planning is simplified as a graph searching problem, that is, to connect the initial and goal configurations to R , and searching R for a path.

Since the Configuration space is represented implicitly, the complexity of sampling-based approaches tends to be dependent on the difficulty of the path, and not so much on the global complexity of the manipulation scenario or the dimension of the configuration space [17, 22]. The advent of these methods has significantly extended the applicability of motion planning methods [16, 18-21]. Typical examples include the Probabilistic Roadmap Planners (PRM) [16, 17, 45-48], single query planners like Randomized Path Planner (RPP) [13], Ariadne's Clew [49-51], Expansive-space Trees (EST) [19, 52-54], and Rapidly-exploring Random Trees (RRT) [21, 22, 55-57].

Environment Property

Judged from the environment property perspective where the robot operation is performed, motion planning methods are roughly classified into the following two types.

- Most of the existing techniques for motion planning are designed for the environment whose information is known prior and unchanged, such as those described for the BMPP problem and its variants, where it is assumed that the geometry and location of the obstacles are completely known and unchanged.
- Some motion planning algorithms have been developed for robot operations in unstructured and dynamic environments, i.e., the environment information is not known in advance, or the environment/task may change during one manipulation cycle [23, 28, 58]. Since industry productions demand more industrial robots to deal with complex operations with unstructured and dynamic environments, this type of motion planners will attract more research attention in future.

Time Mode of Planning

Robot motion planning can be done either off-line or on-line.

- Off-line Planning

Off-line motion planning is a one-shot computation prior to executing any motion. It constructs the plan in advance, based on a known model of the environment, and then hands the plan off to an executor. It requires all pertinent data to be available in advance.

- On-line Planning

In contrast, on-line planning is an ongoing activity that relies on a continuous flow of information about events occurring in the environment. Practically, the primary distinction between off-line and on-line planning is the computation time, i.e., in the latter mode, motions must be both planned and executed fast enough in response to the environmental events [59]. A typical on-line planner can be sensor-based, which interleaves sensing, computation and action.

Among the classifications reviewed above, the cell-decomposition methods, potential field approaches and sampling-based methods are the most commonly used algorithmic approaches.

2.2 Planning in Dynamic Environments

In many industrial or real-life robotic applications, robot arms operate in dynamic environments, i.e., environments with obstacles or changes whose occurrence in time and space is not precisely known ahead of time. The early introductions to motion planning in dynamic environments can be found in [60, 61], where motion planning in the presence of obstacles moving along known trajectories is studied.

Many existing planning methods deal with robot manipulation in *quasi-dynamic* environments. For example, in [50], each required assembly operation of the robot is planned and executed while the environment remains static during one operation cycle. After every assembly operation, the product or the environment will change its geometry. In another example given in [36], the robot grips different objects or changes the tools. Therefore, the kinematic chain of the robot, including the gripped objects, will change in different operation cycles. This type of application in *quasi-dynamic* environments does not strictly demand the

motion planner to possess the on-line computation capability.

In robotic applications, many manipulation tasks involve dynamic environments. For example, for a pick-and-place task on a conveyor belt, moving objects result in a changing environment; for a manipulation task requiring multi-manipulator cooperation, each manipulator becomes a dynamic obstacle for the other robots. More advanced motion planners are required to deal with manipulations in dynamic environments. This type of application requires real-time motion planning, i.e., the planning time should be less than the sampling interval between two consecutive operation instructions. This requirement thus leads to challenges to researchers. There are relatively less research results in this category. The detailed literature survey of this type of motion planners is presented in this section.

2.2.1 Planning Methods Incorporating Time into the Configuration Space

To handle robot motion planning in dynamic environments, some researchers extend the C-space concept by adding a dimension of time, thus leading to a new concept named configuration-time space (CT-space) [62-64]. A point in this CT-space represents a robot state and can be expressed as $(q; \dot{q}; t)$, indicating the configuration, q , the velocity of the robot, $\dot{q}$, and the time, t . Using this new concept, the moving obstacles are mapped to a static forbidden region in this CT-space. A collision-free trajectory is then computed in the CT-space as a curve segment connecting the initial and goal states lying outside the forbidden region. This curve must be time-monotone, i.e., at any time t_j , its tangent must point to the half-space $t > t_j$. In other words, paths in the state space must move forward as time increases. Based on this concept, in [54], a randomized motion planner for robots is proposed to achieve a specified goal under kinematic and/or dynamic motion constraints while avoiding collision with moving obstacles with known trajectories. However, this type of methods only applies when the obstacle trajectories are prior known.

Instead of defining the problem in the CT-space, the Velocity-Tuning method decouples

motion planning into a path-planning part and a motion-timing part [65]. After the path is planned in the conventional C-space, the timing part is then tuned in the second phase. However, the algorithms developed based on this method are not complete. In [66-68], a similar concept named “the velocity obstacle” is proposed. In [69, 70], a local goal-oriented obstacle-avoidance method based on the Non-Linear Velocity Obstacle (NLVO) concept is proposed which takes into account the known or estimated obstacle trajectories in a given time horizon; combining the results with graph-expansion techniques, an incremental global motion planner in a dynamic environment is proposed. Theoretically, when time is incorporated into the state space, the complexity of motion planning is higher than that in the static configuration space. Thus the above concepts have mainly been applied to mobile robots which are relatively less dimensional than typical industrial manipulators.

2.2.2 Reactive Planning

Another direction in robot motion planning in dynamic environment is to further reduce the execution time of those existing planners that have been verified efficient in static environments so as to make them applicable in changing environments. Following this idea, it is possible to make full use of the existing motion planning methods. Reactive planning is a typical example belonging to this category.

Reference [71] proposes a reactive approach for a planar serial manipulator to avoid moving obstacles. The proposed architecture simplifies the problem of path planning for a serial manipulator to that for a master part (usually the tool-tip) of the manipulator. The rest of the parts of the manipulator follow the motion of the master part and adjust their movement in real-time when obstacles are sensed.

In [72], the global and local path planning are combined together to produce an on-line and reactive path planning for a system of multi manipulators. In the off-line stage, employing the concept of Estimation of the Probability of Faults (EPF), the subspace of the configuration space that minimizes the probability of not finding a collision-free path is found. In the

on-line stage, based on local information, the search is executed within the subspace found in the off-line stage. This on-line path planning thus has the advantages of general local approaches, i.e., it is simple and also reacts fast with low computational cost. On the other hand, this approach avoids the drawbacks of the local approaches by using the off-line stage to minimize the probability of finding blockages and escape from local minima. This approach has been tested on a multi-manipulator system of three five-link robots, and high success rate in on-line path planning of two and three robots in pick-and-place operations are reported.

References [23, 73] describe a practical and commercial motion planner named AMROSE, in which the global planning and local planning are organized as follows:

- *Local planner*: named as motion engine, mimicking the molecule dynamics, that is, artificial forces like repulsive forces between obstacles and links, repulsive forces between links, and attractive forces exerted from goals.
- *Global planner*: named as path finder, performing path planning for a point (the tip of end-effector) in Cartesian space. The global path finder decomposes the global motion planning problem into a sequence of local planning problems which can be solved by the local planner. It then identifies a feasible path in terms of a series of via-points leading from one terminal point to the other. Subsequently the path finder builds a 3D binary space partitioning (BSP) representation of the environment and a connectivity graph is constructed. Finally, a searching algorithm is employed in the planner to find the shortest path connecting the two terminal points.

2.2.3 Sampling-Based Roadmap Methods

The sampling-based motion planning approaches often assume that the environment is static. They could be extended to dynamic applications by incorporating the absolute notion of time as an additional dimension in the configuration space, as defined in the aforementioned CT-Space. However, since the obstacle motions are not assumed to be periodic (cyclic), the configuration space is highly transitory. As a consequence, for applications involving a dynamic environment, building a roadmap corresponding to a static environment during a

preprocessing phase is not useful for such configuration spaces [27]. On the contrary, the roadmap needs to be reconstructed or modified in response to the environmental changes.

In an early approach [74, 75], all nodes that become invalid after an obstacle changes are simply deleted. However, the only way to fill the created empty regions is to sample again for new configurations, which is a costly procedure.

In [37, 76, 77], the dynamic roadmaps is constructed, where a road map is pre-built for a manipulator in an obstacle-free Cartesian workspace so that the presence of obstacles can be accounted for by modifying only the affected portion of the road map. Under this dynamic roadmap, the space saving comes from the following aspects:

- a. The C-space is not represented explicitly, instead using the roadmap to denote the connectivity of the free space.
- b. The mapping from the workspace to C-space is encoded making use of the neighbourhoods in the workspace. In detail, the C-space corresponding to some representative cells in the workspace are adopted as references to the C-space of neighbouring cells.
- c. The C-space corresponding to each representative cell is encoded by reducing the contained redundancy.

In [36], dynamic roadmap is employed to deal with a quasi-dynamic environment. In this reference, a grid-based cell decomposition of the workspace, W , and a roadmap R only considering the robot and the static obstacles in W are calculated in the pre-computation phase. The grid G stores in each cell $c \in G$ all nodes and edges of R that are affected by c . During the operation, every time the manipulation task is changed, i.e., the environment is altered, the dynamic obstacles are tracked and the affected nodes and edges of R are updated accordingly. This operation is called roadmap maintenance. However, in cases when large portions of the roadmap become invalid, the roadmap may fail to return a path and the valid nodes and arcs have to be retrieved by single-query planning methods.

In [78], a two-stage method is adopted to deal with moving obstacles. Firstly, a roadmap for the robot and the static obstacles is computed, without considering the presence of moving obstacles. Then, to solve the path query, portions of the roadmap are updated by verifying whether existing edges are collision-free with respect to the current position of the moving obstacles. After invalidating the colliding edges in the roadmap, if a collision-free path can be found in this remnant roadmap, the planner reports success; else, a single-query technique is employed to add nodes and edges until a new path is available. By combining the merits from both PRM type planner and single query planner, plus the “lazy” collision evaluation technique, the proposed approach allows answering queries very quickly when the moving obstacles have little impact on the free-space connectivity.

In [28], aiming for real-time motion-planning of mobile manipulators in dynamic situations, a motion planning paradigm based on problem decomposition is proposed. The overall planning problem is decomposed into two planning subtasks: capturing the connectivity of the low-dimensional workspace using an adaptive wave-front expansion algorithm; then, the solution to this low-dimensional problem is used as a guide to solve the planning problem in the configuration space of the manipulator by PRM method. Since the planning is executed in two subspaces with lower dimension than the overall problem space, a large reduction of the overall complexity can be obtained. In very tight and complicated workspaces, however, it is possible that no path will be found using this framework, even if one exists.

2.2.4 Other Approaches

Evolutionary Approach

Reference [79] addresses the problem of real-time adaptive and trajectory-optimized planning of manipulator motion in an environment with changes that are not known beforehand. A general approach based on genetic operations is introduced, and simultaneous planning and execution of not only collision-free but also optimized motion are reported, taking into account manipulator constraints. Reference [80] presents an evolutionary fuzzy neural-network (FNN) based approach to exploit the redundancy of an industrial manipulator

with seven DOF, named PA-10, to adapt its configuration to avoid moving obstacles in a teleoperated task.

Parallel Approach

Parallel computing makes use of a set of processors to work cooperatively to solve a computational problem [81]. By concentrating the power of multi computational units, parallel computing can implement a mission more efficiently than a single unit. If a motion planning method can be executed in parallel, the same planning problem can be speeded up, depending on the scale of the parallel computing equipment. This provides a useful tool for on-line motion planning. Some typical parallel implementations of robot motion planning are given in [82-86].

2.3 Summary

In concluding the literature review phase of this research work, some observations were arrived at which are summarized below for robot motion planning in a dynamic environment.

- Reactive approaches can respond fast to the environmental changes, thus has the potential to deal with motion planning in dynamic environments.
- To adapt to dynamic changes, the sampling-based methods need to dynamically maintain the roadmap, or pre-compute the connectivity information.
- If possible, parallel implementation can speed up the motion planning process, making it better suitable for on-line planning.

Chapter 3 Motion Planning in Workspace

This thesis investigates the general motion-planning problem for manipulators, which means the planning task is to be performed in a three-dimensional, dynamically changing environment. The dynamic changes in the environment put forward new demands to motion planning, i.e., motions must be both planned and executed fast enough in response to the changes. For this purpose, the workspace (W-space) is chosen as the planning space which is described in this chapter.

For motion planning in the W-space, the inputs are variables defined in the W-space such as the geometry of the robot and obstacles, positional relationships between the robot and the obstacles, etc. Based on these inputs and the kinematics structure of the robot, the W-space planner aims to generate collision-free and optimal motion of the robot. As the most attractive merit, planning in the W-space has low computational complexity which makes it a promising framework for motion planning in dynamic environments. The low computational overhead mainly comes from the simplicity in describing the positional relationship. No matter how complex the geometric structure of the manipulator is, and how many DOF the robot has, the corresponding variables used to describing the positional relationships between the robot and obstacles (e.g. tool-tip position and the gripper orientation), are at most three-dimensional. Furthermore, these positional relationships are directly obtainable, such as from a sensing system; thus, neither additional transformation nor off-line pre-computation is needed as planning is carried out in the configuration space (C-space). This makes motion planning in the W-space a straightforward task.

3.1 Introduction

In this chapter, the following two key techniques are adopted when developing the new motion planner in the W-space.

- The manipulator is regarded as a collection of agents, and each agent is dynamically assigned one behavior from a predefined behavior collection according to the situation

of the moment.

- A fuzzy logic technique is employed to dynamically allocate behaviors to agents, and to perform the behaviors assigned to agents.

The advantages of applying these two techniques for motion planning are explained in the following:

3.1.1 Multi-agent Strategy for Manipulator Motion Planning

The distributed and cooperative robotics was first developed in the field of mobile robotic systems in the late 1980s [87]. Since then, the multi-agent system theory has been established gradually. Maes defined “agent” as “a computational unit that tries to fulfill a set of goals in a complex, dynamic environment” [88]. Generally, a multi-agent system is defined as a system composed of more than two agents and performs the given task by cooperation among agents.

Due to its inherent distributed nature, a multi-agent system has the potential to accomplish some tasks more efficiently than a single robot. For example, a multi robot system can localize the target faster and more accurately. Compared with a centralized system, a multi-agent system can perform more efficiently by parallel computation [89, 90]. Furthermore, a multi agent system introduces redundancy and therefore is fault-tolerant as the independent calculations of the agents will correctly continue even if one or several agent units fail [89].

The concept of multi-agent system can also be extended to motion planning of multi-link robots. In [91], one individual agent controls one joint and evaluates the sensor readings of the corresponding link. This is called a joint agent and this agent is responsible for integrating the sensor data as well as generating the motion for this joint. In [23], a mechanical element or a combination of several elements of a manipulator is considered as an agent, and a robot manipulator is thus considered as a collection of multi agents. By distributing the control of a manipulator to a set of agents, a complex and intelligent motion of a manipulator emerges from the simple motion attempts of the agents [88].

Due to the above advantages, the multi-agent system concept is chosen in this chapter to plan the motion for robot manipulators, i.e., each joint, together with its driven link, of the robot arm is considered as an agent in our path planner design.

3.1.2 Fuzzy Reasoning

Fuzzy logic provides a formal methodology for representing and implementing the human experts' heuristic knowledge and perception-based actions. Using fuzzy logic, the attributes of human reasoning and decision making can be formulated by a set of simple and intuitive IF (antecedent)–THEN (consequent) rules, coupled with easily understandable and natural linguistic representations. The linguistic values in the rule antecedents convey the imprecision associated with the perceptions, while those in the rule consequents represent the vagueness inherent in the reasoning processes. In other words, fuzzy rule-based systems generate actions based on perceptions. Using the fuzzy reasoning strategy and extract heuristic rules from human experience, the need for an analytic model of the process can be avoided, which is especially meaningful when the system involves complicated dynamics. Figure 3.1 shows the typical structure of fuzzy logic.

Figure 3.1 Structure of fuzzy logic

For robotic applications in dynamic environments, the ability to plan the motion or modify the planned motion in real time is of critical importance. Given its inherent advantages, fuzzy reasoning provides a feasible tool for motion planning of manipulators in changing environments. For example, the low computational cost enables it to generate a good reactivity for environmental changes in navigation tasks of mobile robots [32-34, 92-94]. Moreover, an environmental representation at a coarse resolution is adequate for a fuzzy planner; thus the linguistic description in fuzzy logic actually loosens the required accuracy and reduces the effort in environment description [34]. Other than being applied in motion planning of mobile robots, fuzzy logic has also been verified as a complement of the multi-agent strategy in motion planning of simple manipulators. In [58, 95, 96], each link of a planar robot is treated as an individual agent for goal-approaching/obstacle-avoidance. The distance between a link and its nearest obstacle, the difference between the current link configuration and the target configuration are taken as fuzzy inputs to the fuzzy controller; the outputs are motor commands which determine the motion of each agent.

In this chapter, the multi-agent strategy and fuzzy reasoning technique are combined together for motion planning of manipulators in the W-space.

3.2 Planner Design

Different from a multi-agent system in the robot soccer, our multi-agent system is subjected to kinematics constraints between links, that is, a manipulator link cannot move freely in the Cartesian space but can only move within the range constrained by the preceding links. These constraints introduce additional complexity for the communication and cooperation between agents in motion planning. This complexity increases significantly as the DOF of a manipulator increases. Aiming to overcome this difficulty, a hierarchical structure is considered in the planner design.

Raju et al. proposed fuzzy behavior hierarchy [97] to reduce the size of fuzzy rule base, and Tunstel et al. applied this hierarchy for autonomous navigation and control of multi mobile

robots [98, 99], where the overall robot behavior is decomposed into a bottom-up hierarchy of increased behavioral complexity, and the activity at a given level is a function of behaviors at the level(s) below. Similar to Tunstel's work, in this chapter, a hierarchical structure with two levels is designed to implement the multi-agent strategy and fuzzy reasoning for motion planning of manipulators. As shown in Figure 3.2, the higher-level planner is designed for behavior assignment for each agent in the system. The lower-level planner is designed to quantitatively determine the joint speed according to the behavior assigned by the higher-level planner. These two levels are described in detail below.

Figure 3.2 Structure of the designed motion planner with two levels

3.2.1 Higher-level Planner Design

The role of the higher level in the hierarchical structure is to dynamically assign each link an appropriate behavior from the behavior collection.

Behavior Collection

Before the discussion of the role assignment task, definitions of the robot behaviors need to be presented first. Based on the given task of the planner, the behavior collection is defined to be (*goal_approach*, *slide*, *obst_evade*).

Figure 3.3 Illustration of the behavior collection

The *goal_approach* behavior is illustrated in Figure 3.3(a). The manipulator is required to move from its initial configuration I to the final configuration G . Based on the inverse kinematics, this movement can be decomposed into the movement of individual agents, that is, each joint J_i moves from the initial angle θ_{iI} to the goal angle θ_{iG} , with $i = 1, 2, \dots, n$, and n is the number of joints.

The *slide* behavior is illustrated in Figure 3.3(b), where Δ represents an obstacle. Assume that A_i is the rotational axis of Joint J_i and locates in the paper plane; a point p on Link K is assumed in the danger neighbourhood (to be defined by the user) near obstacle Δ . Observed from the figure, the rotation of Joint J_i around A_i causes Link K to move at the plane perpendicular to the paper, causing p to slide over the obstacle Δ . *Slide* thus commands Joint J_i to rotate about A_i , until Link K “slides” away from the danger neighbourhood.

The *obst_avoid* behavior can also be explained using Figure 3.3(b). Consider Joint J_{i-1} , assume that the rotational axis A_{i-1} is perpendicular to the paper plane, and another link, for example Link K , is in the danger zone. This behavior thus commands Joint J_{i-1} to rotate about A_{i-1} so that Link K directly moves away from the danger neighbourhood.

Motion Variables

In the second step of the higher-level planner design, several variables need to be defined describing the motions of the robot links. The detailed definitions are given in Table 3.1.

Table 3.1 Motion variables

Variables	Definitions
<i>IdealSpeed_i</i>	The angular speed Joint J_i should take in next sampling interval in order to approach its goal position, assuming no obstacle
<i>PresSpeed_i</i>	Angular speed of Joint J_i in present sampling interval
The following variables are defined by assuming Link K is around the neighbourhood of an obstacle.	
<i>IdealSpeedDistChange_i</i>	At present sampling interval, if Joint J_i moves at speed $IdealSpeed_i$, while keeping other joints stationery, the distance change between Link K and the obstacle caused by Joint J_i 's movement
<i>PresSpeedDistChange_i</i>	At present sampling interval, if Joint J_i moves at speed $PresSpeed_i$, while keeping other joints stationery, the distance change between Link K and the obstacle caused by Joint J_i 's movement
<i>DistuDistChange_i</i>	<p>Impose a unit disturbance to joint angle θ_i, the distance change between Link K and the obstacle.</p> <p><i>Comments: Sometimes when the angular speed of Joint J_i is very slow or is zero, then the distance change between the obstacle and Link K is very small or is zero. We are thus not able to tell if the motion of Joint J_i will cause collision to Link K according to $PresSpeed_i$. Therefore we purposely introduce a disturbance signal to Joint J_i so that it performs an obvious movement.</i></p>

Fuzzy Variables

In this planner design, a fuzzy logic algorithm is applied to realize the motion planning at the higher level. The fuzzy variables of the higher-level planner are described in Table 3.2. The membership functions of these input variables are illustrated in Figures 3.4(a) - 3.4(d). The membership functions of the output fuzzy variables are selected to be fuzzy singletons (a fuzzy singleton is a fuzzy set whose support is a single point in the universe of discourse with a membership function of one) as shown in Figure 3.4(e).

Table 3.2 Fuzzy variables of the higher-level planner

Input Variables	Output Variables
$Link\ subscript\ difference\ (i-K)^*$	$goal_approach$
$IdealSpeedDistChange_i$	$slide$
$IdealSpeed_i$	$obst_evade$
$DistuDistuChange_i$	

Note*: $1 \leq i$, $K \leq n$, n is the DOF of the manipulator.

Figure 3.4 Membership functions of the higher-level planner

Fuzzy Rules

The fuzzy rules to be implemented in the higher-level planner are defined in Table 3.3. Link K is assumed to be around the neighbourhood of the obstacle.

Defuzzification

In this fuzzy logic algorithm, the Mean of Maximum (MOM) method which calculates the mean of the output values with the highest possibility degrees is employed as the defuzzification mechanism [100]. As mentioned earlier, fuzzy singletons are chosen as the membership functions for the outputs of the higher-level planner, hence, the planner outputs a singleton denoting one behavior.

Notes about the Higher-level Planner

In this higher-level planner design, the following concerns have been taken into consideration.

- (1) The distance between the links and the obstacle must be kept at certain value so as to avoid collision.
- (2) The moving direction of each link is very critical. For example, even though the distance between Link i and the obstacle is beyond the collision threshold (defined by the user), if Link i is to move towards the obstacle in the next sampling interval, collision might occur. The planner must avoid this situation.
- (3) How the position of a link is influenced by its preceding joints must be considered as well. Basically this influence depends on the robot kinematics structure and the current configuration. For example, if Link i receives a motion command, the positions of the outer links of Link i will be changed due to the rotation of joint i . Will these changes cause collision? There must be a means to detect the distance and the change of the distance between the outer links and the obstacle. For this purpose, the variables $IdealSpeedDistChange_i$ and $PresSpeedDistChange_i$ in Table 3.1 are defined to represent the above kinematics relationship in a straightforward manner.

Table 3.3 Fuzzy rules of the higher-level planner

Fuzzy Rule 1	If $i-K > 0$, then Link i is assigned <i>goal_approach</i> behavior
Analysis	$i > K$ means that Link i is the outer link of Link K . In this case, motion of Link i will not effect the obstacle avoidance of Link K .
Fuzzy Rule 2	If $i-K \leq 0$ and if $IdealSpeedDistChange_i$ is N , then Link i is assigned <i>obst_evade</i> behavior
Analysis	$i < K$ means that Link i is the inner link of Link K , and $IdealSpeedDistChange_i = N$ means if Link i moves at the speed of $IdealSpeed_i$, Link K will move nearer to the obstacle. In this case, Joint i should stop or move in the reverse direction.
Fuzzy Rule 3	If $i-K \leq 0$, $IdealSpeedDistChange_i$ is Z , and $IdealSpeed_i$ is NZ , then Link i is assigned <i>slide</i> behavior.
Analysis	Link i is the inner link of Link K . When $IdealSpeedDistChange_i$ is Z and $IdealSpeed_i$ is NZ , it means Link i has not reached its goal position, but whether or not moving Link i at speed of $IdealSpeed_i$, the distance between Link K and the obstacle will not change. In this case, Link i is assigned <i>slide</i> behavior.
Fuzzy Rule 4	If $i-K \leq 0$, $IdealSpeedDistChange_i$ is Z , $IdealSpeed_i$ is Z , and $DistuDistaChange_i$ is Z , then Link i is assigned <i>slide</i> behavior.
Analysis	Link i is the inner link of Link K . When $IdealSpeedDistChange_i$ is Z and $IdealSpeed_i$ is Z , it means that Link i has reached its goal position. $DistuDistaChange_i$ is Z , it means that if a disturbance signal is purposely introduced to joint angle θ_i , the distance between Link K and the obstacle will not change, which means collision will not occur.
Fuzzy Rule 5	For all the other situations, Link i is assigned <i>goal_approach</i> behavior.

(4) The communication between links is also of importance. For example, when Link K goes too near to an obstacle, the higher-level planner will be executed for role assignment and each link will be assigned a role based on the situation occurring; when Link K moves out of the neighbourhood of the obstacle, the higher-level planner then sends the message to all other links for their role reassignment.

3.2.2 Lower-level Planner Design

It is clear by now that the higher-level planner is designed for behavior assignment to each agent in the system. The lower-level planner is designed to quantitatively determine the joint speed according to the behavior assigned by the higher-level planner. This lower-level planner is also developed using fuzzy logic approach. The fuzzy variables of this planner are listed in Table 3.4.

Table 3.4 Fuzzy variables of the lower-level planner

Behaviors	Input Variables	Output Variables
<i>goal_approach</i>	<i>Joint Error</i> <i>Joint Limit</i>	<i>Joint speed for Link i at (t+1)</i>
<i>slide</i>	<i>Distance</i> <i>Joint Limit</i> <i>PresSpeed_i</i>	<i>Joint speed for Link i at (t+1)</i>
<i>obst_avoid</i>	<i>PresSpeedDistChange_i</i> <i>Distance</i> <i>PresSpeed_i</i>	<i>Joint speed for Link i at (t+1)</i>

In table 3.4, the input variable *Joint Error* is defined as the normalized difference between the goal configuration and present configuration, i.e.,

$$JE_i = \frac{\theta_{iG} - \theta_i}{\theta_{iMax} - \theta_{iMin}} \quad (3.1)$$

where θ_i is the present angle of Joint J_i , θ_{iG} is the goal angle of Joint J_i , θ_{iMax} and θ_{iMin} represent the upper and bottom threshold angles of Joint J_i respectively, and $-1 \leq JE_i \leq 1$; another input variable *Joint Limit* is defined as

$$JL_i = \frac{\theta_i - (\theta_{iMax} - \theta_{iMin})/2}{(\theta_{iMax} - \theta_{iMin})/2} \quad (3.2)$$

and $-1 \leq JL_i \leq 1$; the input variable *Distance* is defined as the distance between the link and the obstacle.

The membership functions of these variables are illustrated in Figure 3.5 and the fuzzy rules are described in Tables 3.5, 3.6 and 3.7. The fuzzy rules and the membership functions of the fuzzy variables are similar as those defined in [58, 95, 96]. Note that the sign of the fuzzy output variables is also taken into consideration.

Figure 3.5 Membership functions of the lower-level planner

Table 3.5 Fuzzy rules of *goal-approaching* behavior

		<i>Joint Limit</i>			
		NB	NS	PS	PB
<i>Joint Error</i>	NB	PB	PB	PB	NB
	NS	PS	PS	PS	NS
	NZ	PS	PZ	PZ	NS
	Z	PZ	Z	Z	NZ
	PZ	PS	NZ	NZ	NS
	PS	PS	NS	NS	NS
	PB	PB	NB	NB	NB

Table 3.6 Fuzzy rules of *slide* behavior

<i>PresSpeed_i=N</i>		<i>Joint Limit</i>			
		NB	NS	PS	PB
<i>Distance</i>	Z	PB	NS	NS	NB
	S	PS	NS	NS	NS
	B	PZ	Z	Z	NZ
<i>PresSpeed_i=Z</i>		<i>Joint Limit</i>			
		NB	NS	PS	PB
<i>Distance</i>	Z	PB	NS	PS	NB
	S	PS	NS	PS	NS
	B	PZ	Z	Z	NZ
<i>PresSpeed_i=P</i>		<i>Joint Limit</i>			
		NB	NS	PS	PB
<i>Distance</i>	Z	PB	PS	PS	NB
	S	PS	PS	PS	NS
	B	PZ	Z	Z	NZ

Table 3.7 Fuzzy rules of *obstacle_avoid* behavior

$PresSpeed_i=N$		$PresSpeedDistChange_i$		
		N	Z	P
<i>Distance</i>	Z	PB		NB
	S	PS		NS
	B	PZ		NZ
$PresSpeed_i=P$		$PresSpeedDistChange_i$		
		N	Z	P
<i>Distance</i>	Z	NB		PB
	S	NS		PS
	B	NZ		PZ

3.2.3 More Design Considerations

3.2.3.1 Possible Collision Types

During manipulation, it is necessary to ensure that both the end-effector and its preceding links are collision-free. The possible collision types when one obstacle is considered in the environment can be generalized as follows:

- (1) Single link encounters the obstacle neighbourhood.
- (2) The common joint of two connective links moves into the obstacle neighbourhood.
- (3) The end-effector moves into the obstacle neighbourhood.

Figure 3.6 Possible collision types

All these collision situations are illustrated in Figure 3.6. The proposed motion planner has the ability to handle all these types of collisions. Figure 3.6(a) describes the first situation, where the sensor system reports one link entering into the neighbourhood of the obstacle. In this case, the planner will follow the procedures described in the above sections to perform the operations. For the second situation shown in Figure 3.6(b), assuming that the two links are K and $K+1$ respectively, the obstacle-evade actions of these two links must be considered independently. For Link K , the planner will generate a set of driving commands in the format of joint speed, i.e., $(\dot{\theta}_{1,K}, \dot{\theta}_{2,K}, \dots, \dot{\theta}_{N,K})$. For Link $K+1$, driving commands $(\dot{\theta}_{1,K+1}, \dot{\theta}_{2,K+1}, \dots, \dot{\theta}_{N,K+1})$ can also be generated by the planner. Then, the summation of these two sets of commands, $(\dot{\theta}_{1,K} + \dot{\theta}_{1,K+1}, \dot{\theta}_{2,K} + \dot{\theta}_{2,K+1}, \dots, \dot{\theta}_{N,K} + \dot{\theta}_{N,K+1})$, is used as the driving inputs to the joints. For the third situation given in Figure 3.6(c), the end-effector can be considered as a link; hence, procedures in the first situation will be applied.

3.2.3.2 Integrating a Backtracking Mechanism in Unsuccessful Trials

In this chapter, a back-tracking mechanism is applied when a robot moving trial is unsuccessful. The principle of path searching with such a backtracking mechanism is explained in Algorithm 3.1.

During every sampling interval, the higher-level planner first assigns each joint a behavior; then, the lower-level planner determines the angular speed for each joint. At sampling instant $t_i = t_0 + i * T$, assuming Link K is within the danger zone, based on the fuzzy rules defined in the planner, a proper control command will be generated to drive Link K away from the danger zone. At the next sampling instant $t_{i+1} = t_0 + (i+1) * T$, if Link K is out of the danger zone, then this trial *obst_evade* succeeds; if at t_{i+1} , Link K is still within the danger zone after performing the control command, the behaviors of all agents will remain the same. In this case, a similar control command will be generated to continuously drive Link K . This procedure is repeated until Link K is out of the danger zone.

Algorithm 3.1 Path planning with the Backtracking Mechanism, $BTrack(t_i, K, Cfg_i)$ **Input:** t_i : current time Cfg_i : current robot configuration, $Cfg_i = (q_1, \dots, q_N)$ K : the distance between Link K and obstacles is less than a preset threshold value $d_{\min}$, denoted as $Dist(link_K, obst)|_{t_i} < d_{\min}$ **Output:**Link K gets out of the danger zone within a preset number of iterations, denoted as $Dist(link_K, obst)|_{t_{i+n}} > d_{\min}$, $n < n_{\max}$, or the planner reports fail after trying a certain number of iterations, denoted as $Dist(link_K, obst)|_{t_{i+n_{\max}}} < d_{\min}$.

```

1:       $t \leftarrow t_i$ 
2:      Call higher-level planner;
3:      while ( $i < n_{\max}$ )
4:      {
5:          For  $j=1$  to  $K$ 
6:          {      Call lower-level planner, obtain joint speed  $\dot{q}_j$ ;
7:              Move the robot:  $q_j \leftarrow q_j + \dot{q}_j * T$ ;
8:          }
9:          Call Swift++;    //calculate distance between the robot and obstacles
10:         If ( $Dist(link\_K, obst)|_{t_i} > d_{\min}$ ) return success;
11:         Else                                      $i++$ ;
12:     }
13:     If ( $Dist(link\_K, obst)|_{t_{i+n_{\max}}} < d_{\min}$ )
14:     {      call  $BTrack(t_{i+n_{\max}}, K, Cfg_i)$ ;    }

```

However, it is not guaranteed that Link K will move into the safe zone after a predefined sampling steps. When it happens, this planner will move the robot back to the configuration at time instant t_i , and call the higher-level planner once more to drive the robot towards a different direction. This is referred to as a back-tracking ability. The proposed planner is equipped with this back-tracking mechanism which enables the system to remember the trials it has performed. The robot is thus able to track back to its previous configurations and avoid earlier unsuccessful trials when executing new searching.

3.3 Simulation

To demonstrate its effectiveness, the proposed W-space motion planner is tested in several simulated scenarios. An educational robot named ESHED SCORBOT ER4pc with five DOF and an industrial robot PA-10 with seven DOF are used as the simulation prototypes throughout this thesis. The specifications of these two robots are listed in Appendix A and B respectively.

The program flow of the developed W-space planner is shown in Algorithm 3.2. During simulation, the initial and goal robot configurations are specified and some obstacles are also placed in the manipulation environment. When the motion planning module in this program is executed, the higher-level planner first performs the role-assignment task according to the current environment and the moving trends of each link. Afterwards, the lower-level planners are executed to determine the speed of each joint for the current sampling interval. During the motion planning, if the predefined sampling steps are executed or a preset time duration elapses but the danger link has not moved away from the obstacle, the present moving direction of the danger link is considered invalid for obstacle avoidance, and the back-tracking mechanism will drive the robot to go back to the nearest configuration where it initially encounters the obstacle. Then the motion planner will drive the link to try a different direction.

Throughout this thesis, the computer programs for simulation are written in C++ language; the geometric models of robots and obstacles are firstly constructed in Pro/Engineer Wildfire software, then exported to .obj file format, which are then read into the simulation scenario by C++ programs and visualized via OpenGL; the fuzzy logic module is in c language from Mathworks [101] which is called into the simulation program; the distances between robot links and obstacles, such as Line 7 in Algorithm 3.1, are calculated using the SWIFT++ software developed by University of North Carolina [102]. All simulations are run on a personal computer with 1.5 GHz Pentium-M CPU and 512MB memory under Windows XP operating system.

Algorithm 3.2 On-line path planning in a dynamic environment**Input:** q_{init} : the init configuration q_{goal} : the goal configuration**Output:**A motion sequence of the robot, $\{q(t)|t = t_0 \rightarrow t_1, q_{t_0} = q_{init}, q_{t_1} = q_{goal}\}$

```

1:   $i \leftarrow 0, q_{t_0} = q_{init}$ 
2:  while ( $t_i < t_{max}$  and  $q_t \neq q_{goal}$ )
3:  {    If ( $\forall K \in (0, n], Dist(link\_K, obst)|_{t_i} < d_{min}$ )
4:      Call  $BTrack(t_i, K, Cfg_i)$ ;
5:      Else
6:      {    for  $j=1:n$ 
7:          {
8:              Joint  $J_j \leftarrow goal\_approaching$ , and obtain  $\dot{q}_j$ ;
9:               $q_j \leftarrow q_j + \dot{q}_j * T$ ;
10:         }
11:         If ( $q_t = q_{goal}$ )    return success;
12:     }
13: }
```

3.3.1 Simulation Case Employing the SCORBOT robot: Single Static Obstacle around the Forearm

In this simulation case, a small, static and spherical obstacle is placed around the forearm of the robot. The initial and goal configurations are shown in Figure 3.7. In terms of geometric complexity, the SCORBOT robot prototype is represented by a total of 1544 triangles, and the sphere consists of 100 triangles. The detailed steps of the motion during simulation, as recorded in Figure 3.8, have shown that the manipulator has bypassed the obstacle and reached the goal position successfully. The joint trajectories throughout this simulation case are illustrated in Figure 3.9.

With respect to time performance, we evaluate the motion planners by measuring the cycle planning time in this thesis. Referring to Equation 1.1 in Chapter 1, in the i -th planning cycle at time $t_i = t_{i-1} + T$, the motion planner requires a time duration Δt_i in order to return the joint step change Δq_{t_i} . This Δt_i is the actual cycle planning time and can be measured in

Figure 3.7 Initial and goal configuration of static simulation case 1

Figure 3.8 Motion planning process of the static simulation case 1

each planning cycle during motion planning. In general, the value of the planning interval T is chosen so that $T > \Delta t_i$ since additional works other than planning, like visualization etc., need to be executed in one planning cycle. In this simulation case, it is observed that the cycle time is not more than one millisecond (the time accuracy threshold of the developed simulation program is one millisecond). In other words, an on-line planning cycle can be completed almost instantly.

Figure 3.9 Joint trajectory of static simulation case 1

3.3.2 Simulation Case Employing PA-10 Robot: Single Static Obstacle around the Tool-tip

In this simulation case, a big spherical obstacle is placed in the way of the end-effector to the goal position. The initial and goal configurations are shown in Figure 3.10(a). In terms of geometric complexity, the robot prototype PA-10 is represented by a total of 1840 triangles, and the sphere consists of 100 triangles. The screenshots in Figure 3.10 demonstrate the detailed motion steps of the manipulator in this scenario. From the steps recorded in this figure, it is observed that the robot end-effector has successfully avoided the obstacle and

reached the goal configuration. The corresponding joint trajectories are illustrated in Figure 3.11. During this simulation case, time performance similar to Case 1 is observed, that is, the time to execute a single planning cycle is less than 1 millisecond.

Figure 3.10 Motion planning process of the static simulation case 2

Figure 3.11 Joint trajectory of static simulation case 2

3.4 Conclusions and Discussion

In this chapter, a novel manipulator motion planner based on fuzzy reasoning approach is proposed. The W-space is chosen as the planning space. By considering the robot as a collection of agents, the overall planning problem is distributed to the planning of a set of agents. A hierarchical structure is designed to implement the planner, in which the higher level is applied to dynamically assign each agent an appropriate behavior and the lower-level planner is used to determine joint speeds according to different behaviors assigned. Moreover, a back-tracking mechanism is also integrated into the motion planner, aiming to avoid earlier unsuccessful trials. Simulation studies in two typical static situations have demonstrated that the developed motion planner has low computational requirements and is effective for static obstacle avoidance.

As an initial attempt to develop a practical on-line motion planner, this planner version provides us with useful experience in on-line motion planning. There are several limitations of this planner that we intend to overcome. Firstly, the planner tries to drive the robot links to

approach the goal configuration in a trial-and-error manner, which does not guarantee a successful path finding. In other words, the planner is not able to predict the existence of a collision-free path in a given operation scenario. Thus the motion planner developed in the W-space lacks completeness. To make the planner complete, equipping the planner with the ability of global localization is of prime importance.

Secondly, the back-tracking mechanism is valid and it avoids earlier unsuccessful trials and helps to find a collision-free path as demonstrated in two simulation cases. However, every time the planner tracks back to a previous configuration, the prior trials it had executed will be of no meaning if the environment had changed, which causes this planner version to be only suitable for static working environments.

To accommodate the requirement for on-line motion planning which demands computational efficiency, a new motion planner is needed. Taking the above limitations into design consideration, in the next chapter, a new on-line motion planner with enhanced ability of global localization and ability to deal with dynamic obstacles will be proposed.

Chapter 4 Improved Workspace Planning

In this chapter, the W-space planner will be further improved to achieve enhanced ability of global localization and better performance in avoiding dynamic obstacles.

4.1 The Second Version of the W-space Planner

In the second version of the W-space planner, a new planner architecture will be introduced, in which the robot links/joints are organized into two groups with different tasks allocated to each group. The first group refers to the end-effector, which guides the robot motion in the W-space and a vector-format fuzzy reasoning is proposed for this purpose; the other robot links are allocated into the second group, whose major mission is to avoid collision with obstacles, and a reactive approach resembling the potential field approach is applied for this purpose.

4.1.1 Planner Architecture

In Chapter 3, all links/joints of manipulator are organized into a decentralized system with no priority difference between them, and the tasks of all agents are dynamically allocated and coordinated by a higher-level planner. In contrast, the new W-space planner reorganizes all robot links/joints into two groups based on their respective functions in motion planning.

- *End-Effector - Unit for Motion Guidance*

In real robotic applications, the manipulation tasks are mostly expressed through the movement of the end-effector. For example, in part transfer, the task is specified as “the end-effector carrying the payload from the initial position to the destination”. In arc welding, the task is “the end-effector holding the torch to track the welding line”. Thus the end-effector is assigned the unit for *motion guidance* to represent the specified task requirement in a robot application.

- *Other Links/joints - Units for Obstacle Avoidance*

Once the motion of the end-effector is determined, the joints need to rotate accordingly to realize it. When the robot moves with the end-effector following the planned trajectory, the other robot links also need to be collision-free. Therefore, the duty of these links is to avoid both collision with obstacles and self-collision.

4.1.2 Development of Vector-format Fuzzy Reasoning for the End-effector

In Chapter 3, a basic fuzzy-reasoning technique in scalar format is employed. The inputs to the fuzzy planner, i.e., the distances between obstacles and robot links, and the changes of the distances, are all scalar variables. This treatment is based on the assumption that one joint can only rotate in clock-wise or counter clock-wise mode (one DOF). Since the robot end-effector is assigned as an independent unit to guide the motion of the whole robot in the W-space, its motion can be quite different from that of a single joint. The position of the end-effector tip (EE tip) relative to obstacles is a three-dimensional vector, and the trajectory of the EE tip forms a spatial curve in the W-space.

In general, in Cartesian space, the variables describing the motion of a manipulator end-effector, such as position, velocity of the EE tip and force relative to other objects or coordinate frames, contain both the magnitude and pointing information.

Since the magnitude is a scalar, it is straightforward to be used as a fuzzy input. In contrast, the usage of the pointing information is rather intriguing because information in three dimensions is involved. In existing studies, multi-dimensional vectors are normally decomposed into a set of scalar elements so that the traditional scalar-type fuzzy logic technique can be used to deal with spatial reasoning. For instance, in [103], the position of the gripper tip relative to an obstacle, which is a three-dimensional (3D) vector, is decomposed into three scalar components (θ, ϕ, d) with respect to a spherical coordinate frame, and then these components are “fuzzified” individually. Similarly, for mobile robots, most existing fuzzy planners dealing with planar applications represent the pointing information using

angles (which are again scalar variables) between objects and the robot [32, 92, 93]. A typical example about motion planning of an Underwater Robotics Vehicle (URV) in 3D space is described in [94], where the trajectory is generated in the horizontal and vertical planes separately. In both planes, the position of the URV relative to the coordinate origin is still expressed by a scalar angular variable.

This decomposition approach, however, may lead to problems in certain situations. According to [97], a complete fuzzy rule base has m^n different rules, where m is the number of fuzzy sets in each fuzzy variable, and n is the number of system variables in a rule. For a complex robotic application, we may see that, if we directly employ the common scalar-based fuzzy logic technique after decomposing all the vector-format variables such as position, velocity and force in Cartesian coordinates into scalar components, the number of fuzzy variables, n , needed to be considered simultaneously becomes very large. As a result, we would need to formulate a very large and complex set of fuzzy rules and henceforth a very complex rule base with $O(m^n)$ size, which could become an intractable task. This is especially true for robots operating in dynamic environments.

A multi-dimensional vector-format variable, indeed, can be treated as a unitary fuzzy linguistic variable during the fuzzy reasoning. Such an integrated strategy could make fuzzy propositions more compact and possibly match the natural language expression more efficiently. If multi-dimensional fuzzy variables are reasoned directly, the number of required fuzzy variables and the scale of the rule base can be reduced considerably. For example, if the pointing portion of a position vector is treated directly as a unitary variable in motion planning, then compared with the methods in [94, 103] where two angles are considered separately, one dimension can be saved and the size of the corresponding complete fuzzy set will be reduced from m^2 to m . Therefore, a new fuzzy-logic technique that operates on vector-format fuzzy linguistic variables can be utilized for on-line robot motion planning in 3D dynamic environments. To implement this new approach, a set of new membership function is defined, and a series of new vector-based fuzzification, fuzzy inference and

defuzzification procedures are established. By handling the multi-dimensional pointing as a unitary vector, common behaviors of a robot such as goal approaching and obstacle avoidance can be mapped in fuzzy reasoning conveniently, which significantly reduces the complexity of the fuzzy logic approach for 3D motion planning.

4.1.2.1 Mathematical Description of the Vector-format Fuzzy Logic Reasoning

The general procedure of the proposed vector-format fuzzy logic motion planner is similar to that of common fuzzy logic controllers, and consists of the following main components: 1) fuzzifier; 2) rule base; 3) inference engine (approximate reasoning); and 4) defuzzifier. In what follows, we present in detail the development of the new fuzzy motion planner, and highlight the unique characteristics of the vector-format fuzzy reasoning through comparison with existing scalar-format fuzzy reasoning.

Fuzzifier

The fuzzifier module transforms the measured crisp data into a fuzzy set. The membership function for a 3D vector can be obtained by extending the membership function defined for scalar type variables. We consider a typical membership function for a scalar fuzzy variable, e.g., Gaussian type function,

$$f(x, \sigma, c) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-c)^2}{2\sigma^2}} \quad (4.1)$$

where σ is the standard deviation and c denotes the mean value. Based upon the above mathematical expression, the multi-dimensional Gaussian function can be obtained as in [104] as

$$g(x, y, z) = A e^{-\frac{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2}{2\sigma^2}} \quad (4.2)$$

where A is a constant, σ is again the standard deviation, x_0 , y_0 , z_0 are mean values along the x , y and z axes respectively. Assuming $\bar{P}$ as a spatial vector defined in the 3D space, the following normalization procedure leads to the pointing information,

$$\bar{P}_1 = \frac{\bar{P}}{\|\bar{P}\|} = (p_x, p_y, p_z), \text{ i.e., } p_x^2 + p_y^2 + p_z^2 = 1 \quad (4.3)$$

where p_x, p_y, p_z are the components of the unit vector $\bar{P}_1$ at x, y , and z directions respectively.

From Equations (4.2) and (4.3), the Gaussian membership functions for the 3D unit pointing vector $\bar{P}_1$ can be defined. The details are listed in Table 4.1. In this table, a total of six fuzzy members are defined. For example, $\bar{P}_U = (0, 0, 1)$ indicates the pointing “UP” in a pre-defined Cartesian coordinate, $z_{\bar{P}_1}$ is the projection of $\bar{P}_1$ on the z -axis. If the pointing is absolutely “UP”, then $F_U(\bar{P}_1)$ returns a firing strength “1”. The firing strength usually falls within $[0, 1]$ following the Gaussian distribution. Other membership functions are defined in the similar manner. Figure 4.1 shows the graphical representation of the pointing membership function “UP”. As shown in Figure 4.1(b), when we observe the xz cross-section, the firing strength $F_U(\bar{P}_1)$ of $\bar{P}_1$ is represented by the distance away from the sphere surface. Using these fuzzy membership functions, the normalized pointing vector $\bar{P}_1$ can be mapped into a fuzzy set. Furthermore, the projection of the above fuzzy membership “UP” on the xz cross-section can be transformed into a normal 1D Gaussian distribution “up” shown in Figure 4.1(c). In other words, the defined multi-dimensional memberships can be regarded as generalizations of the normal 1D memberships to 3D cases. When we define θ as the angle between $\bar{P}_1$ and the x axis, the members “FRONT”, “UP”, “BACK” and “DOWN” will correspond to $\theta = 0^\circ, 90^\circ, 180^\circ$ and 270° , respectively, in the xz cross-section. In other words, the universe of discourse of the equivalent scalar angle θ is divided into 4 fuzzy sets. Similarly, when we define ϕ as the angle between $\bar{P}_1$ and the x axis in the xy cross-section, the projections of the members “FRONT”, “RIGHT”, “BACK” and “LEFT” will correspond to $\phi = 0^\circ, 90^\circ, 180^\circ$ and 270° , respectively. This sets up a conversion approach between the vector-format fuzzy reasoning and the traditional decomposition method. Combining these membership functions with the existing scalar-based fuzzification method for distance, we may then develop fuzzy reasoning for the entire position information.

Table 4.1 Membership functions for linguistic variable - orientation

Fuzzy Membership Name	Membership Function Expression
UP	$F_U(\bar{P}_1) = \begin{cases} \exp[-\frac{ \bar{P}_1 - \bar{P}_U ^2}{2\sigma^2}] & z_{\bar{P}_1} \geq 0 \\ 0 & z_{\bar{P}_1} < 0 \end{cases}, \quad \bar{P}_U = (0, 0, 1)$
DOWN	$F_D(\bar{P}_1) = \begin{cases} \exp[-\frac{ \bar{P}_1 - \bar{P}_D ^2}{2\sigma^2}] & z_{\bar{P}_1} \leq 0 \\ 0 & z_{\bar{P}_1} > 0 \end{cases}, \quad \bar{P}_D = (0, 0, -1)$
LEFT	$F_L(\bar{P}_1) = \begin{cases} \exp[-\frac{ \bar{P}_1 - \bar{P}_L ^2}{2\sigma^2}] & y_{\bar{P}_1} \leq 0 \\ 0 & y_{\bar{P}_1} > 0 \end{cases}, \quad \bar{P}_L = (0, 1, 0)$
RIGHT	$F_R(\bar{P}_1) = \begin{cases} \exp[-\frac{ \bar{P}_1 - \bar{P}_R ^2}{2\sigma^2}] & y_{\bar{P}_1} \geq 0 \\ 0 & y_{\bar{P}_1} < 0 \end{cases}, \quad \bar{P}_R = (0, -1, 0)$
FRONT	$F_F(\bar{P}_1) = \begin{cases} \exp[-\frac{ \bar{P}_1 - \bar{P}_F ^2}{2\sigma^2}] & x_{\bar{P}_1} \leq 0 \\ 0 & x_{\bar{P}_1} > 0 \end{cases}, \quad \bar{P}_F = (1, 0, 0)$
BACK	$F_B(\bar{P}_1) = \begin{cases} \exp[-\frac{ \bar{P}_1 - \bar{P}_B ^2}{2\sigma^2}] & x_{\bar{P}_1} \geq 0 \\ 0 & x_{\bar{P}_1} < 0 \end{cases}, \quad \bar{P}_B = (-1, 0, 0)$

(a)

(b)

(c)

Figure 4.1 Graphical illustration of the orientation membership function “UP” ($\sigma = 0.3$)

Fuzzy Rule Base

Similar to the scalar-format fuzzy planner, the vector-format fuzzy motion planner also uses the “IF.....THEN.....” rules. The main difference is that now the input/output variables are in vector format. Two examples are shown as follows, with $\bar{P}_1$ being the input and $\bar{P}_2$ the output.

Rule example 1

IF the current pointing $\bar{P}_1$ of an obstacle is RIGHT, THEN the moving direction $\bar{P}_2$ of the robot is UP.

Rule example 2

IF the current pointing $\bar{P}_1$ of an obstacle is UP, THEN the moving direction $\bar{P}_2$ of the robot is FRONT.

Inference Engine

In a fuzzy inference engine, the conclusion is inferred from the IF-THEN rules. When there are multiple rules in the rule base, the final value of the inference engine is obtained by taking a union or intersection operation, depending on the implication approach, over all the results of the individual rules. This can be illustrated through the following example.

Inference engine example:

For an example of a fuzzy planner, we assume its rule base contains two rules listed in the previous section of the “Rule Base”. Let the current input vector be $\bar{P}_0 = (1, 2, 3)$, and the pointing vector of the robot position is then obtained as,

$$\bar{P}_1 = \frac{\bar{P}_0}{|\bar{P}_0|} = \frac{(1, 2, 3)}{\sqrt{1^2 + 2^2 + 3^2}} = (0.2673, 0.5345, 0.8018)$$

Assuming that $\sigma = 0.5$, for Rule 1, the firing strength of output $\bar{P}_2$ is obtained based on the membership function defined in Table 4.1,

$$F_R(\bar{P}_1) = \exp\left[-\frac{|\bar{P}_1 - \bar{P}_R|^2}{2\sigma^2}\right] = \exp\left[-\frac{(0.2673-0)^2 + (0.5345+1)^2 + (0.8018-0)^2}{2*0.5^2}\right] = 0.1554$$

If Mamdani Implication [100] is adopted, we have $\mu_{UP'} = \min\{F_R(\bar{P}_1), \mu_{UP}\}$, where $\mu_{UP'}$ and μ_{UP} locate on the unit spherical surface which represents all possible values that the pointing vector may take (Figure 4.2(b)).

Figure 4.2 Graphical representation of the fuzzy inference for rule 1 ($\sigma = 0.5$)

For Rule 2, the firing strength of output $\bar{P}_2$ can be obtained in a similar manner,

$$F_U(\bar{P}_1) = \exp\left[-\frac{|\bar{P}_1 - \bar{P}_U|^2}{2\sigma^2}\right] = \exp\left[-\frac{(0.2673-0)^2 + (0.5345-0)^2 + (0.8018-1)^2}{2*0.5^2}\right] = 0.4525$$

We then have $\mu_{FRONT'} = \min\{F_U(\bar{P}_1), \mu_{FRONT}\}$, here $\mu_{FRONT'}$ and μ_{FRONT} are also located on the unit spherical surface. For a clear demonstration, the implication processes of Rule 1 and Rule 2 are illustrated in Figures 4.2 and 4.3 respectively.

Figure 4.3 Graphical representation of the fuzzy inference for rule 2 ($\sigma = 0.5$)

The final step in the inference engine is to combine the results of these two rules. Here we take the union operation as $\mu_{\bar{P}_2} = \max\{\mu_{UP'}, \mu_{FRONT'}\}$, which is illustrated in Figure 4.4.

Figure 4.4 The union of the two rules

Defuzzifier

Defuzzification is a process of mapping from a fuzzy control/planning action to a crisp control/planning action. There are two common defuzzification methods for the existing scalar-based fuzzy reasoning, i.e., the Center of Area (*COA*) method and the Mean of Maximum (*MOM*) method [100]. Since the *MOM* method would fail to capture the information in the entire distribution range of the linguistic variables, the *COA* method is preferred as the default defuzzification method for the vector-format fuzzy reasoning approach. We consider a discrete case. The crisp value accumulated from a spherical surface under the *COA* method is given as

$$\overline{Z}_{COA} = \frac{\sum_{i=1}^n \mu_{\overline{p}_i} \cdot \overline{p}_i \cdot A_i}{\sum_{i=1}^n \mu_{\overline{p}_i} \cdot A_i} \quad (4.4)$$

where n is the number of elements in the fuzzy set, $\overline{p}_i = (p_{ix}, p_{iy}, p_{iz})$ is the normalized 3D position vector which locates on the spherical surface, and A_i is the area cell. Then as shown in Figure 4.5, the following spherical coordinate expression of $\overline{p}_i$ is appropriate to represent the defuzzification process.

$$\begin{cases} p_{ix} = \sin \phi \cos \theta \\ p_{iy} = \sin \phi \sin \theta \\ p_{iz} = \cos \phi \end{cases} \quad (4.5)$$

where $0 \leq \theta \leq 2\pi$ and $0 \leq \phi \leq \pi$. Assume $\Delta\theta$ and $\Delta\phi$ are the step lengths of θ and ϕ respectively. The position vector $\overline{p}_i$ can be further expressed as

$$\overline{p}_i = (\sin \phi_k \cdot \cos \theta_j, \sin \phi_k \cdot \sin \theta_j, \cos \phi_k)$$

where $0 \leq j \leq J, 0 \leq k \leq K$ and $J = 2\pi / \Delta\theta$, $K = \pi / \Delta\phi$. We also have

$$A_i = (\sin \phi_k \cdot \Delta\phi) \cdot \Delta\theta$$

Figure 4.5 Illustration of the spherical coordinate ($|\overline{P}_i| = 1$)

Substitution of Equation (4.5) into Equation (4.4) leads to

$$\overline{Z}_{COA} = \frac{\sum_{k=1}^K \sin \phi \cdot \Delta \phi \left(\sum_{j=1}^J \mu_{p_i} \cdot (\sin \phi_k \cdot \cos \theta_j, \sin \phi_k \cdot \sin \theta_j, \sin \phi_k) \cdot \Delta \theta \right)}{\sum_{k=1}^K \sin \phi \cdot \Delta \phi \left(\sum_{j=1}^J \mu_{p_i} \cdot \Delta \theta \right)} \quad (4.6)$$

Consider a continuous case where $J \rightarrow \infty$ and $K \rightarrow \infty$. In this case, the COA method leads to

$$\overline{Z}_{COA} = \frac{\int_0^\pi \sin \phi \cdot d\phi \left(\int_0^{2\pi} \mu_{\overline{P}(\theta, \phi)} \cdot (\sin \phi \cdot \cos \theta, \sin \phi \cdot \sin \theta, \cos \phi) \cdot d\theta \right)}{\int_0^\pi \sin \phi \cdot d\phi \int_0^{2\pi} \mu_{\overline{P}(\theta, \phi)} \cdot d\theta} \quad (4.7)$$

For the original illustrative example that is a discrete case, we apply Equation (4.6) in the defuzzification. The output vector is thus obtained as $\overline{P}_{2_{COA}} = (0.3467, 0.0092, 0.3107)$. After unitization, we have $\overline{P}_{2_{COA}} = (0.7445, 0.0197, 0.6673)$, which is shown in Figure 4.6. It is worth emphasizing that in the above example of fuzzy reasoning, we deal with the “pointings” as unitary linguistic variables directly, which leads to a compact fuzzy rule set.

Figure 4.6 The output vector after defuzzification

4.1.2.2 Applying the Vector-format Fuzzy Approach to End-effector Navigation

To verify its feasibility and efficiency, we apply the vector-format fuzzy approach to guide the motion of the robot end-effector in the W-space in this section. For this purpose, a behavior-based fuzzy planner with dynamic weight allocation is designed based upon the proposed vector-format fuzzy reasoning. As shown in Figure 4.7, this fuzzy planner is similar to that in the first W-space planner version in respect of structure; on the other hand, the three basic behaviors in the current planner version run in parallel while only one behavior is selected to run at certain moment in the first planner version.

Figure 4.7 The two-level fuzzy planner in vector-format for navigation of the end-effector

4.1.2.2.1 Basic Behaviors of the End-effector

At the lower level of the planner, the following three behaviors are designed to model the

basic motion of the end-effector.

goal_approach

This behavior always leads the EE tip towards the goal position.

obst_evade

The idea behind this behavior is that when the obstacle gets close to the end-effector, the latter will reactively move away from this obstacle.

move_around

The purpose for this behavior is that in 3D space, if the direction from the current position of the EE tip towards its goal is blocked by a nearby obstacle, the end-effector should take the nearest journey to move around the obstacle. As illustrated in Figure 4.8, the dashed area refers to a wall, which blocks the way of the EE tip (E), C is the center of this wall, and G denotes the goal position. In fact, this situation is a typical case of local minima.

A local minimum is a commonly encountered problem in motion planning, and causes the robot to be trapped in configurations other than the goal. Normally it occurs when the planning algorithm is based on local information [15]. The current W-space planner version makes use of the obstacle information around the links; hence, it is in principle a local planner and is not free of local minima.

Many researches have tried to solve the local minima problem. For example, in [105], a disturbance signal is introduced to attract the robot to move away from the local minima; in [13], an arbitrary motion is introduced to escape the local minimum; in [106, 107], a potential field free of local minimum is built; in [108], a arbitrary sub-goal is generated if the search of the planner encounters a deep local minimum well; in [109], a number of sub-goals in the discretized free C-space are randomly generated, followed by parallel searching mechanism to find a path connecting the initial and the goal configurations via the sub-goals. Though these approaches are effective, there is no approach available to achieve both effectiveness and efficiency in terms of local-minimum escaping in on-line motion planning.

For the situation shown in Figure 4.8, the *move_around* behavior is particularly applied to

enables the robot to move around the wall. By this behavior, the EE tip is commanded to move along direction $\overrightarrow{Poi_A}$, which is parallel to the connecting line $\overline{IC}$, with I being the intersection point of line $\overline{GE}$ with the wall. This behavior takes into account the shape and the size of obstacles.

Figure 4.8 The direction for the end-effector to move round an obstacle

In practical robotic applications, the obstacle shape and dimensions can be detected by proximity or distance sensors such as ultrasound or laser range finder. We may introduce a variable $Size_Obst$ to characterize the size of an obstacle being large or small. If the size is considered large, the EE tip needs to take more deviation in order to go round the obstacle.

Inputs

Inputs to the fuzzy motion planner implementing the basic behaviors include: the distance ($Dist_O$) and the pointing ($\overrightarrow{Poi_O}$) extracted from the position of an obstacle with respect to the EE tip, the distance ($Dist_G$) and the pointing ($\overrightarrow{Poi_G}$) extracted from the goal position of the EE tip, the error between $\overrightarrow{Poi_G}$ and $\overrightarrow{Poi_O}$ which is defined as $\overrightarrow{E_{Poi_GO}} = \overrightarrow{Poi_G} - \overrightarrow{Poi_O}$, and $Size_Obst$ which denotes the equivalent radius of an obstacle.

In Table 4.2, six membership functions, UP, DOWN, FRONT, BACK, LEFT and RIGHT are defined for the vector type inputs $\overline{Poi_O}$ and $\overline{Poi_G}$. For $\overline{E_{Poi_GO}}$, membership function S(small) is defined with constant reference value (0, 0, 0) and

$$F(S) = \exp\left[-\frac{|\overline{E_{Poi_GO}} - \overline{P_s}|^2}{2\sigma^2}\right], \quad \overline{P_s} = (0, 0, 0)$$

Three membership functions N(near), M(medium) and F(far) are designed for the scalar variables $Dist_O$ and $Dist_G$. Three membership functions S(small), M(medium) and B(big) are designed for the scalar variable $Size_Obst$.

Outputs

Output variables of the lower fuzzy motion planner include $\overline{Poi_M}$ and Mag_M . Here, $\overline{Poi_M}$ is a 3D vector denoting the pointing which the EE tip should follow according to the current environmental information. The same membership functions of $\overline{Poi_O}$ and $\overline{Poi_G}$ are designed for $\overline{Poi_M}$. Mag_M is a scalar variable representing the magnitude of the translational change, and three linguistic labels S(small), M(medium) and B(big) are designed for it. The translational motion generated in the behaviors of *goal_approach* and *obst_evade* is expressed as $Mag_M * \overline{Poi_M}$, and the motion from *move_around* is $Mag_M * \overline{Poi_A}$.

Among the inputs and outputs described above, the typical triangle or trapezoid membership functions like those defined in Section 3.2.2 are designed for scalar variables, and the Gaussian type of memberships proposed in this chapter are employed for vector-type variables.

Fuzzy Rules

Three groups of rules are defined, each corresponding to one behavior. The rule base is shown in Table 4.2.

Table 4.2 Fuzzy rule base

Group1- goal-approach	Inputs		Outputs	
	$Dist_G$	$\overline{Ori_G}$	Mag_M	$\overline{Ori_M}$
	N		S	
	M		M	
	F		B	
		UP		UP
		DOWN		DOWN
		FRONT		FRONT
		BACK		BACK
		LEFT		LEFT
		RIGHT		RIGHT
	Inputs		Outputs	
Group2- obst_evade	$Dist_O$	$\overline{Ori_O}$	Mag_M	$\overline{Ori_M}$
	N		B	
	M		M	
	F		S	
		UP		DOWN
		DOWN		UP
		FRONT		BACK
		BACK		FRONT
		LEFT		RIGHT
		RIGHT		LEFT
	Inputs		Outputs	
Group3- move_around	$Size_Obst$	$\overline{E_{Ori_GO}}$	Mag_M	
	S	NS*	S	
	S	S	M	
	B	NS	M	

	B	S	B		
Group4- Dyn-weight	<i>Inputs</i>		<i>Outputs</i>		
	<i>Dist_O</i>	$\overline{E_{Ori_GO}}$	<i>w1</i>	<i>w2</i>	<i>w3</i>
	N		S	B	
	M		M	M	
	F		B	S	
		S			B
		NS			S

* NS =nonsmall.

Rules in Group 1 are developed for the *goal_approach* task. For example, the first rule in Group 1 is given as:

If *Dist_G* is N, then *Mag_M* is S.

This rule can be explained as follows. “*Dist_G* is N” means that, at the current moment, the EE tip is close to its goal position, thus the EE tip should move at a small speed towards the goal position. For another example, the fourth rule in Group 1 is given as:

If $\overline{Poi_G}$ is UP, then $\overline{Poi_M}$ is UP.

This rule can be explained as follows. “ $\overline{Poi_G}$ is UP” means that, at the current moment, the goal is at the “UP” direction, thus the EE tip should follow this direction to move towards the goal.

Rules in Group 2 are developed for *obst_avoid*. For example, the first rule is given as:

If *Dist_O* is N, then *Mag_M* is B.

This rule can be explained as follows. “*Dist_O* is N” means that, at the current moment, the end-effector is close to at least one obstacle, thus the end-effector should move away from this obstacle at a large speed. For another example, the fourth rule in this group is given as:

If $\overline{Poi_O}$ is UP, then $\overline{Poi_M}$ is DOWN.

This rule can be explained as follows. “ $\overline{Poi_O}$ is UP” means that, at the current moment,

the obstacle is located at the “UP” direction with respect to the end-effector, thus the EE tip should move towards the “DOWN” direction against the obstacle.

Rules in Group 3 are developed for the *move_around* task. For example, the first rule in Group 3 is given as:

If *Size_Obst* is S, and $\overline{E_{Poi_GO}}$ is S, then *Mag_M* is M.

In this rule, “ $\overline{E_{Poi_GO}}$ is S” is used to denote the proposition “the direction toward the goal is blocked by obstacle”. Alternatively, this proposition can be interpreted as “IF $\overline{Poi_G}$ equals $\overline{Poi_O}$ ”, where $\overline{Poi_O}$ acts as a fuzzy value of $\overline{Poi_G}$. However, $\overline{Poi_O}$ keeps changing during the process, thus the error between these two ‘pointings’, $\overline{E_{Poi_GO}} = \overline{Poi_G} - \overline{Poi_O}$, is adopted, which has the constant reference value (0, 0, 0) for membership function S(small). Therefore, this rule can be explained as follows. “*Size_Obst* is S” means that the obstacle located between the end-effector and the goal is of small size. “ $\overline{E_{Poi_GO}}$ is S” means that an obstacle is located at a similar direction with the goal position of the EE tip; and then, the end-effector should take a medium deviation along the direction of $\overline{Poi_A}$.

It can be observed that through the design of the above fuzzy sets, the fuzzy propositions relating to the pointing contain six rules in both group 1 and group 2, and three rules in group 3. As stated earlier, if the pointing is considered in the traditional separate fashion, i.e., decompose the pointing into θ and ϕ , then both θ and ϕ need 4 fuzzy sets to achieve the same accuracy; thus, $4^2 = 16$ rules are needed to implement the same reasoning relating to the pointing in each group. The proposed vector-format fuzzy reasoning thus reduces the scale of the rule base considerably.

4.1.2.2.2 Dynamic Weights Allocation

In order to coordinate the above three basic behaviors of the end-effector effectively, their

fuzzy weights are allocated dynamically according to the situation at the current sampling instant. The logic of the weight allocation can be expressed by the following rules.

- If the neighbourhood of the end-effector is clear, then we fully run *goal_approach*.
- If there are obstacle(s) near the end-effector, then we assign the *obst_evade* behavior with high weight.
- If the direction of the EE tip towards the goal is blocked by obstacle(s), then we assign *move_around* behavior with a high weight.

The above logic is summarized into the fourth group of fuzzy rules in Table 4.2.

Using $VF_i()$ to denote the basic behaviors implemented by the proposed vector-format fuzzy planner, the position change rate of the EE tip can then be expressed as

$$\begin{aligned}\dot{X}_E &= \sum w_i * VF_i() \\ &= w_1 * VF_1() + w_2 * VF_2() + w_3 * VF_3()\end{aligned}\quad (4.8)$$

Also note that this fuzzy planner is designed to manage the motion of the end-effector, which is a single unit; while the hierarchical planner in Chapter 3 is used to manage all links/joints of a manipulator.

4.1.3 Reactive Approach for Links to Avoid Obstacles

When the robot joints rotate accordingly to drive the end-effector to follow the trajectory found by the vector-format fuzzy planner, the robot links may get close to the obstacles. To avoid such type of potential collisions, the robot must respond quickly to move the links away from the obstacles.

The potential field method meets this requirement by computing a virtual force F to indicate a direction and a magnitude of force acting on the robot caused by a nearby obstacle. This force F can then be translated into joint torque using the Jacobian at configuration q of the robot. This method can effectively map the low-dimensional force vector F from the workspace into the higher-dimensional joint space of the manipulator. In this research, since only the robot kinematics is considered, an alternative approach based on the scalar-format fuzzy logic and

general-inverse Jacobian, which resembles the potential field method, is adopted.

Figure 4.9 explains the idea of how a link evades a nearby obstacle by repulsive positional change, where p is a point on the k -th link, q is a point on the nearest obstacle, and $|\overline{pq}|$ denotes the minimum distance between the k -th link and the obstacle at the current moment. The robot joints should rotate properly so that point p moves along $\overline{pq}$ and away from the obstacle, and the magnitude of this movement should be inversely proportional to the distance $|\overline{pq}|$. Following this idea, firstly a scalar-format fuzzy logic method is used to generate the required magnitude value; then the inverse-kinematics relation, i.e., the inverse Jacobian equations, is applied to transform the positional change of the danger link into the step changes of all joints. In general, this obstacle avoidance approach works in a reactive manner.

Figure 4.9 The effect of the repulsive evading of a link away from an obstacle

The input to the fuzzy planner is the distance $|\overline{pq}|$, and the output is the magnitude of a positional step change of point p in link k , denoted as Mag . Denoting the fuzzy reasoning in scalar format as $SF()$, we have

$$Mag = SF(|\overline{pq}|) \quad (4.9)$$

As shown in Figure 4.10, the most often used triangular and trapezoid membership functions are adopted for both the input and output fuzzy variables, and the fuzzy rules are shown in Table 4.3. The displacement change rate of point p can then be expressed as

$$\dot{X}_p = Mag * \frac{\overline{pq}}{|\overline{pq}|} = SF(|\overline{pq}|) * \frac{\overline{pq}}{|\overline{pq}|} \quad (4.10)$$

Figure 4.10 Membership function of input and output linguistic variables

Table 4.3 Fuzzy rules

Distance	Positional Step Change
Z	B
S	M
M	S
B	Z

* The rule is read as:

IF Distance is Z, THEN Positional Step Change is B.

And the memberships have the following meanings:

Z: zero

S: small

M: middle

B: big

4.1.4 General-inverse Jacobian

During the on-line path planning under the current W-space planner design, at each planning cycle the vector-format fuzzy reasoning system will output a positional step change (velocity) of the EE tip, $\dot{X}_E$. To realize this velocity, the corresponding joint speeds need to be derived. This is known as an inverse-kinematics problem. For this purpose, the Jacobian matrix with respect to the EE tip, noted as J_n , is introduced here. The process to calculate the general-inverse Jacobian, J_n^+ , of the two robot prototypes are given in Appendix C. Then the corresponding joint change rate can be expressed as

$${}^1\dot{\theta} = (\dot{\theta}_1, \dots, \dot{\theta}_n) = J_n^+ * \dot{X}_E = J_n^+ * \sum_{i=1}^3 (w_i * VF_i()) \quad (4.11)$$

Similarly, after a positional change rate of point p in the k -th link is reported for repulsing the link away from the nearest obstacle, the corresponding joint change rate can be derived by the general-inverse Jacobian. The nearest point p in the k -th link is chosen as the reference point to describe the velocity state of this link. We assume point p has the local coordinate (p_x, p_y, p_z) with respect to the k -th link frame F_k . The Jacobian matrixes of J_p^+ of the two robot prototypes when $k=1, \dots, n$ are also given in Appendix C. Then the joint change rate for obstacle avoidance can be expressed as

$${}^2\dot{\theta} = (\dot{\theta}_1, \dots, \dot{\theta}_i) = J_p^+ * \dot{X}_p = J_p^+ * (SF\left(\left|\frac{\vec{pq}}{pq}\right|\right) * \frac{\vec{pq}}{pq}) \quad (4.12)$$

Then the total joint change rate at time t_i is

$$\dot{\theta} = {}^1\dot{\theta} + {}^2\dot{\theta} = J_n^+ * \sum (w_i * VF_i(Dist, \overline{Poi})) + J_p^+ * (SF\left(\left|\frac{\vec{pq}}{pq}\right|\right) * \frac{\vec{pq}}{pq}) \quad (4.13)$$

4.1.5 Simulation

Applying the new planner architecture and incorporating the vector-format of fuzzy reasoning

and general-inverse Jacobian, the structure of the second version of the W-space planner is shown in Figure 4.11. Since we study the motion planning in dynamic environments where changes may occur continuously, the environment needs to be sensed in each planning cycle and the updated environmental information is then fed to the motion planner. As shown in Figure 4.11, the components located within the dashed frame refer to the planning actions in a single planning cycle. In such a planning cycle, both the module of end-effector navigation and the module of obstacle avoidance are executed once, then the joint changes are summed up. The above cycle is executed recursively until the preset destination is reached.

Figure 4.11 Structure of the second version of W-space planner

In the Figure, the subscript i means the related values are sampled in the i -th planning cycle, $\dot{X}_E(t_i)$ is the velocity of the EE tip generated from the vector-format fuzzy navigator, $\dot{X}_p(t_i)$ is the change rate of the reference point p in the k -th link within the danger zone. The outputs ${}^1\dot{\theta}(t_i)$ and ${}^2\dot{\theta}(t_i)$ are the joint change rate calculated from $\dot{X}_E(t_i)$ and $\dot{X}_p(t_i)$ by applying the general-inverse Jacobin respectively, and $\dot{\theta}(t_i)$ is the total joint change rate in the i -th planning cycle. Simulation studies have been executed to verify the effectiveness of this planner.

4.1.5.1 Case 1 - Environment Containing Single Dynamic Obstacle

This simulation case is designed to test the capability of the planner to deal with the environmental changing. The start and goal configurations are shown in Figure 4.12(a) and Figure 4.12(h) respectively, and one spherical obstacle moves to and from between the start and the goal of the end-effector. Figure 4.12(h) also gives the ultimate trajectory of the EE tip, which clearly shows the evading behavior of the end-effector. Figure 4.13 illustrates the joint trajectories of this simulation case, where avoiding behavior can be observed at $t \approx 26s$ for the 2nd, 3rd and 4th joints. During this simulation case, similar time performance with that in the first W-space planner design is achieved, that is, the execution time of a single planning cycle, including executing the vector-format fuzzy module and the obstacle-avoidance module, is less than 1 millisecond.

Figure 4.12 Motion sequence of simulation case containing a single dynamic obstacle

Figure 4.13 Joint trajectory VS time of the 1st simulation case

4.1.5.2 Case 2 - Environment Involving a Large Wall

This simulation case is designed to test the planner performance in moving around a large wall which is a typical case of local minimum. The start and goal configurations are shown in Figure 4.14(a) and Figure 4.14(b) respectively, and a large static wall is placed between the start and the goal position of the end-effector. As the motion sequence in Figure 4.14 shows, the robot successfully goes around the large wall. Figure 4.15 illustrates the joint trajectories during the planning process. In this simulation case, similar time performance with that in simulation case 1 was observed.

Figure 4.14 Simulation scenario with a large scale wall

Figure 4.15 Joint trajectory VS. time of the 2nd simulation case

4.1.5.3 Case 3 - Complex Scenario Involving Non-convex Obstacles

A more complex simulation case was devised and applied to further test the performance for local-minimum escaping. A static scenario named *Star* from [110] is adopted here. As shown in Figure 4.16, this scenario contains a narrow start and goal space with a broad space between them.

Unfortunately, in this simulation case, the vector-format fuzzy navigator was not able to guide the end-effector out of the start position. The failure suggests that, to escape this type of local minimum, the planner should be improved with a stronger ability of global path finding.

Figure 4.16 Simulation scenario experiencing navigator failure guiding end-effector out of non-convex start position

4.1.6 Summary of the Second Version of the W-space Planner

In summary, with a compact fuzzy rule base, the second version of the W-space planner achieves a better performance in dealing with dynamic environmental changes than the first planner version and is verified to be able to solve a typical local minimum case with a large wall. However, it is still a trial-and-error type of approach and needs to be further improved for global localization. It also focuses on the algorithmic aspects, and employs the EE tip as the reference point to represent the end-effector position but omits the orientation of the end-effector. To suit general industrial robotic tasks, the orientation of the end-effector needs to be taken into account. These issues will be addressed in the next W-space planner version.

4.2 The Third Version of the W-space Planner

In the third version of the W-space planner, utilizing the wrist-decompose structure of typical industrial robots, the overall planning problem is decomposed into the position and orientation parts. In the position sub-problem, the global searching idea is applied to find a path for the wrist center rather than the trial-and-error approaches of the previous two versions. To realize this searching method effectively, in each planning cycle, the W-space is represented employing a hierarchical octree structure. Within this hierarchically represented W-space, an optimal path connecting the current position of the wrist center and its

destination is searched by applying the well-known A* algorithm. In the orientation sub-problem, the robot joints outside the wrist rotate to adjust the gripper orientation locally.

4.2.1 Wrist-decouple Structure and Decomposition Strategy

The majority of industrial robots have such a wrist structure that the last two or three joint axes intersect at the wrist centre. As a result, the inner links below the wrist serve to position the center of robot wrist, whereas the joint axes outside the wrist determines the gripper orientation. For example, the two robot prototypes have such a wrist-decouple structure and Figure 4.15 shows their wrist center positions. In Figure 4.17(a) is the SCORBOT ER4pc robot, whose positioning joints include θ_1 , θ_2 , θ_3 and orientation joints are θ_4 and θ_5 . If we define I_p and I_o as the collection of positioning joints and orientation joints respectively, then for the SCORBOT robot, $I_p = \{1, 2, 3\}$, $I_o = \{4, 5\}$. In Figure 4.17(b) is the Mitsubishi PA-10 manipulator, and $I_p = \{1, 2, 3, 4\}$, $I_o = \{5, 6, 7\}$ based on its kinematics.

(a) ESHED SCORBOT-ER4pc robot with 5 DOF

(b) Mitsubishi PA-10 Robot with seven DOF

Figure 4.17 The wrist-decompose structure of typical industrial robots

A natural way for kinematics analysis of such types of industrial robots is to separate the position part away from the orientation part [63]. Such an idea of decomposing is also of great benefit to robot motion planning. After decomposing the overall planning problem into sub-problems with less DOF, the planning complexity can be reduced considerably. It is noted that in [82] the primitive Configuration-space maps are calculated respectively for the first three positioning links and the three-dimensional gripper of a PUMA robot; in [28], the motion planning of a mobile manipulator is decomposed into planning the movement of the mobile base on the plane and adjusting the manipulator mainly to avoid local obstacles. On the other hand, this decomposition strategy makes a trade-off between the efficiency and the planning completeness, that is, there is no guarantee that a path will be found even if it does exist. Fortunately, there is usually a relative wide clearance between the robot and obstacles in typical industrial robotic applications, so it is possible to reduce this negative effect as much as possible.

Following [82], in this planner version we also decompose the motion planning problem of such industrial robot types into two sub-problems: planning the positional movement of the wrist centre which involves those inner links between the base and the wrist, and adjusting the gripper orientation by those joint axes outside the wrist. Such a planning strategy makes full use of the wrist-decouple structure and reduces the complexity of motion planning by distributing it to two sub-problems with lower DOF.

4.2.2 Planning Position Movement of the Wrist Center

The first sub-problem, i.e., planning the positional movement by inner joints, will be implemented by continuously searching a collision-free path for the wrist center in the Cartesian space. This path will be a time-varying spatial curve in the Cartesian space and can be expressed as

$$X_w = X_w(t_0 + iT), i = 0, 1, 2, \dots \quad (4.14)$$

where T is the planning period. Based upon the above analysis, a fast global-searching algorithm like the well known A*, Dijkstra's algorithm and etc. suits the requirement well.

The path searching can benefit a lot from a hierarchical representation of the planning space due to the following facts. First, a hierarchical space representation such as the well known octree can save storage space considerably. Furthermore, within an octree, a path can be searched starting from the lowest resolution; as a result, the found collision-free path will be an optimal solution which guarantees as large a clearance as possible between the wrist center and the obstacles. Since there is usually a relative wide clearance between the industrial robot and obstacles in typical industrial robotic applications, it is very possible that a path can be found at a low resolution. Therefore, in this third planner version, the planning space will be represented by an octree which is updated with time period T , and the searching algorithm will be executed in this hierarchical space with automatic resolution adjustment. The last step of this sub-problem is to transform the movement of the wrist center into joint changes, which

is typically an inverse kinematics problem.

Following the trajectory found by the searching algorithm, the wrist center can be guaranteed safe at all times; however, this is not true for the inner links. In other words, the inner links may get close to and may even collide with dynamic obstacles. In order to overcome this possible situation, the reactive method as that described in Section 4.1.4 will be employed for those inner links to quickly respond to and move away from possible obstacles getting close to them.

4.2.2.1 Representing the W-space by An Octree

The well-known octree structure is a hierarchical representation of a three-dimensional space or object Ω . In an octree, each node is a cuboid cell which is labeled as *Black*, *White* or *Grey* respectively. Only those *Grey* nodes may have children and each node further down has eight children [4]. Compared with normal discretized representations, the hierarchical octree structure can save memory storage by merging a set of neighbour *White* cells or neighbour *Black* cells to a single *White* or *Black* cell at the upper level. Further, it is easy to change the resolution of the representation in the overall space or locally [111]. Thus it is for these reasons that the octree is widely used in robot motion planning. For instance, Faverjon applies octree (where $m = 3$) to represent a C-space with three DOF [40]; Hayward [112], Kitamura et al [41, 113] employ the octree to represent the W-space for collision detection and path searching.

Figure 4.18 shows a typical octree example from [113], where the original and decomposed geometric entity is given in Figure 4.18(a) and (b) respectively, and the corresponding octree representation is given in Figure 4.18(d) based on the convention in Figure 4.18(c). As shown in Figure 4.18(d), each octree node is labeled with an index value, referred to as *Morton code* [114], which maps three-dimensional coordinates to one-dimensional scalars. A *Morton code* can be obtained by binary interleaving of the coordinates [115], and it has one or several digits, called *directional digit*. The *Morton code* specifies the node position in an octree in the

following manner.

- A node can be tracked by starting from the root, traversing downwards and following its directional digit at each depth.
- At the longitudinal direction, counting from the left-most cell, the decimal value of the Morton code denotes the sequential position of the corresponding cell in the current depth.

An octree representing the space Ω can be obtained by recursively decomposing its cells into eight equal cubic octants until some preset resolution is reached [42].

(a) The original example geometry

(b) The example geometry after decomposition

(c) The ordering of octants

(d) Representation of the example octree with Morton codes

Figure 4.18 The octree shape representation with bit encoding

Implementing an Octree by Bit Encoding

Several techniques are available to store the octree nodes in a computer program.

- a) *Pointer*: It is convenient to represent an octree using pointers by allocating eight pointers per node, each of which points to one of its children. For example, in [116], a

pointer is employed to denote an octree structure with memory requirement of 50 bytes for a single node.

- b) *Linear Octree*: This method assigns a unique key to each black leaf node. This key encodes both the location and the size of the node [117]. The location part is denoted by the *Morton code*, and the size information is denoted by the depth value. Compared with the pointer representation, the linear octree provides more memory savings.
- c) *Array in bit format*: By this method, two arrays are used to represent an octree. Figure 4.19 shows such a representation of the octree in Figure 4.18(d). Both arrays contain 8^L bits corresponding to the L-th depth of the octree, and the value of each bit distinguishes the property of a cell at the current level. In the *Black/white* array, a bit value 0 represents that the node is *White*, and a bit value 1 means the node is *Black* or *Grey*. In the *Leaf* array, a bit value 1 denotes the node has no child, and a bit value 0 for leaf node.

Depth	Black/white	Leaf
0	1	1
1	11101011	01000001
2	$\underbrace{00000000}_1, \underbrace{10111000}_2, \underbrace{00000000}_3, \dots, \underbrace{11111010}_8$	$\underbrace{00000000}_1, \dots, \underbrace{00000000}_7, \underbrace{01000000}_8$
3	$\underbrace{00000000}_1, \underbrace{00000000}_2, \dots, \underbrace{00000000}_{63}, \underbrace{10111010}_{64}$	$\underbrace{00000000}_1, \underbrace{00000000}_2, \dots, \underbrace{00000000}_{63}, \underbrace{00000000}_{64}$

Figure 4.19 The complete representation of the octree in Figure 4.18(d)

Table 4.4 Storage requirement of three octree implementations

Scheme	Theoretical Space requirement (in bytes)	Actual space to represent the octree in Figure 4.18(d) (in Bytes)
Pointer structure	$> 5*(1+n_g)*8$	>160 (normally a pointer need 4 bytes of memory)
Linear octree	$n_b*(3*h+\log(h+1))/8$, [115]	25
Array in bit format	$2 * \sum_{i=1}^h 8^{i-1}$	146

(n_g - number of grey nodes in the tree, n_b - number of black nodes in the tree, h - the maximum tree depth)

The space requirements of these three techniques are summarized in Table 4.4. Based on Table 4.4, although scheme *c* is not the most space-saving solution, it stores all nodes of the tree and thus is most suitable for free neighbour searching. As stated in [115], when free neighbours of a reference cell need to be found, the basic idea for both scheme *a* and *b* is first to ascend the octree until a common ancestor is located and then descend the octree in search of the neighbouring node. Thus, in order to find a neighbour, those cells located at the path from the reference cell to the nearest common ancestor and the path from the common ancestor to the neighbour cell must be visited. In contrast, by scheme *c*, the neighbour nodes can be directly found by mapping the array subscript, and the status of nodes can be judged from the corresponding bit values. As verified later in this chapter, this property of scheme *c* helps to achieve fast on-line path searching. Furthermore, accuracy can still be guaranteed after adopting scheme *c*, at least for typical small-sized and middle-sized industrial robots. For example, assuming a robot workspace with a dimension of $1m \times 1m \times 1m$ and applying scheme *c*, 100 Mbytes of storage space enables the octree to reach a maximum depth of $\log_8(100 \times 8 \times 10^6) \approx 10$, which is equivalent to accuracy of $1000/2^{10} \approx 1mm$ in the Cartesian space. In summary, the octree representation scheme *c* can achieve fast path searching and guarantee accuracy with an acceptable expense in storage space. Thus scheme *c* is chosen in this research.

4.2.2.2 On-line Path Searching

In each planning cycle during the on-line path planning, once the environment is updated, the octree representation will be also updated accordingly as described in the previous section and path searching will be executed in the updated space to find a new collision-free path of the wrist center. When the searching succeeds, a feasible path consisting of a list of consecutive free cells will be obtained, then the robot moves along this on-line path. To implement the above task effectively, an efficient searching algorithm making use of the octree hierarchy to improve the computational efficiency is required.

Neighbour Searching in An Octree

In an octree, when path searching reaches a free cell, the searching algorithm will check all free neighbours for this cell and choose one from them as the next free cell. There are several

existing variations, including finding vertex-, edge-, or face-type neighbours. In this chapter, the face-type neighbours are considered. The diagonal neighbours, which only share a single vertex or edge with the current cell, would result in inflexible paths which clip obstacle corners [115], these are not considered in this study. Since the free cells may be located at different depths, they may have different sizes, which complicates the process of neighbour searching. In [40], one type of recursive searching procedure is applied to find free neighbours in an octree, which represents a three DOF C-space. In [116], an octree is also employed to represent the physical space of an aerial robot. In this chapter, under the octree implementation by arrays in bit format, a neighbour-searching approach similar to that in [40] is adopted and implemented by bit operations, as explained in the following paragraphs.

In an octree, if diagonal neighbours are not admitted, the neighbours of a random cell may be located at any of the six directions, i.e., the positive and negative directions of the x , y and z axes, which are denoted as x_+ , x_- , y_+ , y_- , z_+ , z_- respectively. As an example, Table 4.5 lists the procedure to find all these directions by bit operations, with respect to the node 75 in Figure 4.18. In Table 4.5, column 1 shows the Morton code of the reference cell, and in column 2 is the bit expression of this Morton code. The bit values along coordinates x , y and z are separated and shown in column 3. Column 4 lists the possible bit value of neighbours, that is, plus or minus 1 to the bit value along the x , y and z axes respectively. Column 5 shows the bit interleaving operation, which leads to six directions with bit value and Morton code given in column 6 and 7 respectively. In Table 4.5, there are two exceptions, that is, for $x_+ = 1\ 0\ 0$ and $z_+ = 1\ 0\ 0$, the corresponding neighbours in the octree will be located outside of the space boundary, thus no valid neighbour exists at these two directions.

Table 4.5 Procedures to find the directions for free neighbour searching

Reference node			Free neighbours			
Morton Code	Bit value	Bit value along coordinate	Possible bit value of neighbours	Operation	Bit value	Morton code
75	111 101	x=1 1	$x_- = 1 0 *$	$x_- \oplus y \oplus z **$	111 001	71
			$x_+ = 1 0 0$			
		y=1 0	$y_- = 0 1$	$x \oplus y_- \oplus z$	101 111	57
			$y_+ = 1 1$	$x \oplus y_+ \oplus z$	111 111	77
		z=1 1	$z_- = 1 0$	$x \oplus y \oplus z_-$	111 100	74
			$z_+ = 1 0 0$			

* With respect to cell 75, $x_- = x-1 = 11 - 1 = 10$. The same meaning applies to x_+ , y_+ , y_-

**Here $\oplus$ denotes the bit interleaving operation

As mentioned earlier, after applying the octree representation scheme c , there is no need to find a common ancestor when neighbours are looked for. In this chapter, an approach for free-neighbour searching in a top-down fashion is applied. At each possible direction, the availability of free neighbours is checked from depth 1 till depth $d_{\max}$, with $d_{\max}$ being the maximum depth of the octree. Figure 4.20 demonstrates the searching process to find all free neighbours of the reference cell 75 in the example octree.

Figure 4.20 Searching free neighbours in the octree sample

As an example, the free neighbours at the first searching direction x , i.e., the direction with Morton code 71 in Table 4.5, can be found as follows: at depth $L=1$, the directional digit is 7, check Figure 4.20(b) and the node with index 7 at depth 1 is grey, go down to depth 2; at depth 2, the node with index of 71 is also grey, continue to go downwards to depth 3; recall that the current searching direction is x , and the cells at this depth have smaller size than the reference node 75; thus, among the eight children of node 71 at depth 3, only those with bit value 1 at x direction, i.e., nodes 714(111 001 100), 715(111 001 101), 716(111 001 110) and 717(111 001 111), are neighbours of the reference cell. Check Figure 4.20(a) again, cells 714 and 716 are black while nodes 715 and 717 are white, thus nodes 715 and 717 are valid free neighbours.

In Figure 4.20(b), the white leaf nodes at the terminals of dashed lines denote the required free neighbours of node 75. The validity of these neighbours can be verified in the original space in Figure 4.20(a). The above neighbour-searching process is summarized in Algorithm 4.1 and 4.2.

Algorithm 4.1 Free neighbour searching in an octree, *Search_FN()*

Input: $C(L_c)$: free reference cell at depth L_c $d_{\max}$: the maximum depth of the octree**Output:** $FN(C(L_c))$: free neighbours of $C(L_c)$

```

1:  FN ← ∅
2:  for i=1 to 6
3:    {    L ← 1;
4:        while (L ≤ Lc)
5:          {    call Algorithm 4.2;  }
6:          L = Lc + 1;
7:        while (L ≤ dmax)
8:          {
9:            for j=1 to 2m-1
10:           {
11:             Call Algorithm 4.2;
12:           }
13:         }
14:  }
```

Algorithm 4.2 White/Black/Grey Detection

Input: $M(i, L)$: a reference cell with known *Morton code* and depth i : the i -th searching direction**Output:***White/Black/Grey*: status of the node

```

1:  Extract the Morton code  $M(i, L)$  of the  $i$ -th searching direction with respect to the
    reference cell at depth  $L$  *;
2:  Switch  $M(i, L)$ 
3:    Case White
4:      FN ← FN ∪  $M(i, L)$ ;
5:      Return white;
6:    Case Black
7:      Return black;
8:    Case Grey
9:      L ← L + 1;
```

* The Morton code $M(i, L)$ at Line 2 can be extracted by bit shift and addition.

In the above pseudo-codes, two types of free neighbours, i.e., the free neighbours with larger or the same size and the free neighbours with smaller size, are recognized and searched respectively.

Global Searching Algorithm

There are many widely used searching algorithms, among which the A* algorithm is most suitable in case that a shortest path is desired [35]. The A* algorithm tries to reduce the total number of states explored by incorporating a heuristic estimate of the cost to reach the goal from a given state [118]. This algorithm is widely used in motion planning. For example, in [26], it is employed for path searching in roadmap; in [40], it is used to search non-collision path in the C-space of a robot with three DOF which is represented by an octree. In this chapter, the A* algorithm is also employed to search optimal paths connecting the initial and goal position of the wrist center in the W-space. During path searching, path scoring is the key to determine which node to use when figuring out the path. The following evaluation function is adopted in this chapter:

$F = G + H$, where

- G = the cost to move from the starting point A to a given node in the W-space, following the path generated to there.
- H = the estimated movement cost to move from that given node in the W-space to the final destination, point B.

Multi-resolution Path Searching

Based on the hierarchical nature of the octree representation, multi-resolution searching strategy can provide significant advantages to path planning. For example, in [29, 119], the global planner first searches promising portions of the C-space at a coarse resolution. It then increases the resolution to a finer value if a solution is not found at the current depth, and this finer resolution is only used in promising portions of the C-space. In [38], a quadtree is used to represent the workspace for path planning of a mobile-robot, where a path including grey leaf nodes is first planned in a reduced resolution; subsequently, the path is developed inside the grey nodes in the second stage. Unfortunately, in this method, when a path including grey

nodes is found as unfeasible at a later stage, the planner needs go back to the level with the lowest resolution and re-plan the path. Too many re-plannings will reduce the efficiency of the planner.

In this study, the multi-resolution strategy is also applied to plan the position movement of the wrist center. The A* searching is executed only within free nodes, thus eliminating possible re-planning as occurring in [38]. The planner structure is shown in Figure 4.19, in which the inner loop denotes the actions of the multi-resolution path searching in a single planning cycle. Starting from depth 1, the octree representing the whole W-space is decomposed first, then A* searching is executed within this octree under the current resolution. If no collision-free path is available, the algorithm goes to the next depth and repeats the procedure of decomposing and searching. This loop repeats until a global collision-free path is found or the highest resolution is reached but no path is found. Since the searching carries further only when the path is not found at the current resolution, on-line computational time and memory can thus be saved. The result of the above searching procedure can be expressed as

$$\begin{cases} Path(t_i, \lambda) = A^*(X_i, X_g, Env) \\ X_w(t_i) = Path(t_i, \Delta) \end{cases} \quad (4.15)$$

where X_i and X_g denotes the initial and goal position respectively, and Env refers to the environment. The collision-free path $Path(t_i, \lambda)$ found at time t_i is indexed by the unitized distance traveled along the trajectory, denoted as λ , and satisfies $Path(t_i, 0) = q(t_i)$ and $Path(t_i, 1) = q_{goal}$. We assume a small positive constant Δ to represent the step length along the trajectory and let $Path(t_i, \lambda)|_{\lambda=\Delta}$ be the new position of the wrist center, $X_w(t_i)$. To move the wrist center from the current position $X_w(t_{i-1})$ to $X_w(t_i)$, the corresponding changes of the positioning joints need to be derived, which can be solved by inverse-kinematics. Since the planning of wrist centre involves only the displacement change, the position part of the robot Jacobian matrix, J_w , is thus used and the changes of the positioning joints can be expressed as:

$${}^1\Delta\theta_p = ({}^1\Delta\theta_{p1}, \dots, {}^1\Delta\theta_{pW}) = J_W^+ * (X_W(t_i) - X_W(t_{i-1})) = J_W^+ * (Path(t_i, \Delta) - X_W(t_{i-1})) \quad (4.16)$$

where J_W^+ is the general-inverse of J_W . The Jacobian equations of J_W and J_W^+ for the SCORBOT and PA-10 robots are listed in Appendix C.

4.2.2.3 Obstacle Avoidance of Positioning Links

Applying the same approach for obstacle avoidance as described in Section 4.1.4, the movement of the positioning joints for obstacle avoidance is expressed as

$${}^2\dot{\theta}_p = \left({}^2\dot{\theta}_{p1}, \dots, {}^2\dot{\theta}_{pW} \right) = J_p^+ * \dot{X}_p = J_p^+ * (SF(\left| \begin{matrix} \vec{pq} \end{matrix} \right|) * \frac{\vec{pq}}{\left| \begin{matrix} \vec{pq} \end{matrix} \right|}) \quad (4.17)$$

where p refers to the point which is closest to obstacles in the k -th link ($k \in I_p$) and the Jacobian matrix can be referred to Appendix C.

Summarizing the sub-problem of position planning, the joint step change in the i -th planning cycle at $t_i = t_0 + i * T$ can be expressed as

$$\Delta\theta_p = {}^1\Delta\theta_p + {}^2\dot{\theta}_p * T = J_W^+ * (Path(t_i, \Delta) - X_W(t_{i-1})) + J_p^+ * (SF(\left| \begin{matrix} \vec{pq} \end{matrix} \right|) * \frac{\vec{pq}}{\left| \begin{matrix} \vec{pq} \end{matrix} \right|}) * T \quad (4.18)$$

4.2.3 Planning the Gripper Orientation

At the i -th planning cycle, after the first sub-problem is accomplished and a displacement change of the wrist centre $\Delta\theta_p$ returns, the robot gripper still needs to adjust its orientation for two reasons: firstly, to achieve the specified tool configuration so as to cater for certain task requirement; secondly, to avoid possible collision in cases where obstacles move too close to the gripper. This forms the second sub-problem. Regarding the motion of the gripper with respect to the wrist centre, it is equivalent to the motion planning for a 3-DOF manipulator with fixed base in the changing environment $R(3)$. Many approaches are available to handle this sub-problem. For example, the sampling method, the potential field method like that applied in [28].

In this research, two behaviors will be executed simultaneously corresponding to the missions of the gripper. For the first behavior, one simple solution is to increase the outer joints with a constant rate till the goal orientation is achieved. The corresponding change rate of the outer joints can then be expressed as

$${}^1\dot{\theta}_{Oj} = c(\theta_{dj} - \theta_{0j}) / \left(\sum_{j \in I_O} (\theta_{dj} - \theta_{0j})^2 \right)^{1/2} \quad (4.19)$$

where c is a constant denoting the magnitude of the joint change rate, θ_{dj} and θ_{0j} is the desired and initial position of the j -th joint respectively.

To avoid obstacles, the same reactive method used for the inner links is applied to the outer joints. It is noted that this second behavior can also work well when the robot gripper catches some payload, if the payload is considered as rigidly connected to the gripper. Also, these two behaviors are coordinated depending on the situation with such a rule that when the clearance between the gripper and obstacles is small, a high weightage will be allocated to the second behavior and low weightage to the first behavior; and vice versa. The corresponding joint changes can be expressed as

$${}^2\dot{\theta}_O = \left({}^2\dot{\theta}_{O(W+1)}, \dots, {}^2\dot{\theta}_{O_n} \right) = J_p^+ * (SF(\left| \vec{pq} \right|) * \frac{\vec{pq}}{\left| \vec{pq} \right|}) \quad (4.20)$$

where J_p^+ is the general-inverse Jacobian of the k -th link ($k \in I_O$).

The result of the orientation sub-problem in the i -th planning cycle at $t_i = t_0 + i * T$ can be summarized as

$$\dot{\theta}_O = {}^1\dot{\theta}_O + {}^2\dot{\theta}_O = c(\theta_{dj} - \theta_{0j}) / \left(\sum_{j \in I_O} (\theta_{dj} - \theta_{0j})^2 \right)^{1/2} + J_p^+ * (SF(\left| \vec{pq} \right|) * \frac{\vec{pq}}{\left| \vec{pq} \right|}) \quad (4.21)$$

4.2.4 Simulation

The structure of the third version of the W-space Planner is described in Figure 4.21, and the

meanings of the included variables are similar to those in Figure 4.11. The on-line planning proceeds like this: in the i -th planning cycle, the environment is sensed and then the octree subdivision and path searching is executed following the multi-resolution strategy, until a collision-free path, $path(t_i, z)$, which connects the current wrist center position to the destination is searched out. Afterwards, the corresponding change of inner joints, ${}^1\Delta\theta_p$, is derived based upon the found path; on the other hand, the module of reactive obstacle-avoidance generates the joint change rate ${}^2\dot{\theta}_p$ of the inner joints. The change rate of the outer joints is the summation of a linear increase ${}^1\dot{\theta}_O$ and the output from the reactive obstacle-avoidance module ${}^2\dot{\theta}_O$. The above procedures will be executed recursively until the destination is reached.

In our simulations, the following method is applied to obtain the updated octree representation of the W-space. Initially, the obstacles are represented in surface models, i.e., a collection of triangles. A pre-processing is executed to obtain a collection of points which are uniformly distributed on the triangular surfaces, as shown in Figure 4.22. Later in each planning cycle of the on-line process, firstly the positions of these points are updated according to the motion of the obstacles; afterwards, these updated points will be used as control vertexes to subdivide the octree recursively.

Three typical cases are tested to verify the performance of this planner design.

4.2.4.1 Retesting the “*Star*” Scenario

To verify the new W-space planner design, the static scenario *Star* from [110], in which the second version W-space planner failed to operate, is retested. As shown in Figure 4.23, to find a collision-free path, a finer discretization is needed during the search. During the path searching, the fixed robot base is regarded as an equivalent static obstacle since the end-effector cannot penetrate the robot base.

Figure 4.21 The structure of the third version of the W-space planner

Figure 4.22 Pre-processing the obstacle surfaces to obtain control vertexes

Figure 4.23 Motion sequence of the static simulation case

In terms of geometric complexity, the SCORBOT robot prototype is represented by a total of 1544 triangles, and 7791 control vertexes located on the obstacle boundaries are used to subdivide the planning space into an octree representation. Since this simulation scenario is static, the A* searching is executed only once and reports a collision-free path for the wrist center at depth=5. The time consumed in this planning case, including the time for octree decomposition and A* searching, is about 20 milliseconds. As some intermediate configurations shown in Figure 4.23, following the collision-free path reported by the planner, the robot successfully moves out of the initial trap, passes by the broad space and ultimately arrives at the goal within the other trap. The joint trajectories of this simulation test case are recorded in Figure 4.24.

Figure 4.24 Joint trajectory of static simulation case

4.2.4.2 Environment Containing Multiple Dynamic Spheres

In this case, the proposed planner needs to deal with a complex changing environment. As shown in Figure 4.25, the SCORBOT robot is again employed as the prototype; a static wall is arranged between the robot start and goal configurations, and several moving spheres are also placed in the environment. These spheres are given arbitrary initial velocities and will

rebound elastically when they hit the boundary of the robot workspace, thus they keep wandering in the workspace except that they are not allowed to hit the robot base, as such type of collision is not avoidable for the robot. It is worth noting that the trajectories of the spheres are un-known in advance, so their positions need to be updated at each planning cycle and fed to the planner. Accordingly, in each planning cycle the planner does not know the future movement of the obstacles in advance and only makes use of the obstacle position information at the current moment. This is to simulate the running mode of a general robotic application involving a dynamic and unpredictable environment, where a sensing system is normally equipped to capture the environmental information. In terms of geometric complexity, a total of 4483 control vertexes on the obstacle surfaces are used to subdivide the workspace.

During this simulation test case, the octree corresponding to the static obstacles, noted as *Oct1*, is calculated once and remains unchanged, whereas, the octree corresponding to the dynamic obstacles, noted as *Oct2*, is reconstructed in each planning cycle following the multi-resolution strategy. Starting from the lowest resolution at $L=1$, *Oct2* is decomposed, then the same depth of *Oct1* and *Oct2* are superposed to obtain the W-space representation at the current moment, noted as *Oct12*, followed by the A* path searching in *Oct12*. In equation,

$$Oct12 = Oct1 + Oct2 \quad (4.22)$$

The above action of decomposition-superposition-searching is recursively executed in a single planning cycle until a path is found out.

In Figure 4.25, there are two types of grey cells shown in the scenario, i.e., the grey nodes in which the dynamic balls are located and the grey nodes occupied by the static obstacles. These grey nodes will be larger if the path is searched out at a rougher resolution, and vice versa. As demonstrated by the motion sequence shown in Figure 4.25, throughout this planning case, the robot succeeds in searching its destination, getting around the static wall and keeping a safe distance away from the dynamic spheres.

Figure 4.25 Screenshots of the dynamic simulation case with multi spheres

The results of this simulation test case are given in Figure 4.26. In Figure 4.26(a) shows the joint trajectories with respect to the planning cycle number. Obvious avoiding action is observed at the trajectories of the second and the third joint. Figure 4.26(b) shows the on-line execution time of this simulation case, where the '×' points refer to the time required for octree subdivision and A* path searching in a single cycle, and those '○' points correspond to the time needed to execute the action of obstacle avoidance in a single cycle. The result shows that a total of 333 planning cycles are executed, among which the maximum time duration to find a collision-free path for the wrist center in a single planning cycle is not more than 10 milliseconds; also shown in Figure 4.26(c), all planning cycles are implemented under a octree depth of 4. It is also observed that executing one cycle of obstacle avoidance needs a time of less than 1 millisecond. Such a prominent on-line performance proves the efficiency of the adopted workspace hierarchy and the multi-resolution strategy.

(a) Joint trajectory VS cycle number

(b) Cycle planning time VS cycle number

(c) Ending depth number VS cycle number

Figure 4.26 Time performance of the simulation case 2

4.2.4.3 Simulation with a 7-DOF Robot Prototype

In this case, the Mitsubishi PA-10 robot with seven DOF is used as a prototype to verify the planner efficiency for redundant manipulators, and the same simulation environment with that in Section 4.2.4.2 is employed.

In terms of geometric complexity, the robot prototype PA-10 is represented by a total of 1840 triangles. As demonstrated by the motion sequence in Figure 4.27, throughout this planning case, the robot succeeds in searching its destination, getting around the static wall and keeping a safe distance away from the dynamic spheres. The simulation results including the joint trajectories, the time duration of each planning cycle and the depth number under which each planning cycle ends are given in Figure 4.28.

Figure 4.27 Dynamic simulation case

(Note that in this scenario, the white cells refer to the grey nodes where the static wall and robot base are located, the yellow boxes denote the grey nodes in which the dynamic balls are located, and the blue cells are the collision-free path for the wrist center)

(a) Joint trajectories VS cycle number

(b) Cycle planning time VS cycle number

(c) Ending depth number VS cycle number

Figure 4.28 Simulation result of case 3

4.2.5 Conclusion and Discussion of the Third Version of the W-space Planner

The third version of W-space planner makes full use of the robot wrist-decouple structure and reduces the complexity of motion planning by distributing to the position part and the orientation part, both with lower DOF. The former sub-problem is solved by searching collision-free paths for the wrist centre in the Cartesian space, and the latter sub-problem is implemented by locally adjusting the gripper orientation. Simulations verify that this new planner version achieves enhanced performance in goal finding in dynamic environments.

Some limitations are recognized during the planner development, which will be solved in our future work.

1) *Incompleteness*: It is noted that the proposed planner makes a trade-off between completeness and efficiency. As a result, sometimes it cannot find a path even if it exists.

2) *Collision-free path of the wrist centre may not be applicable for the robot links*: In the example shown in Figure 4.29, there exist many collision-free paths for the wrist centre connecting positions I and G; however, no collision-free path exists for the whole robot due to

the fact that the obstacle is located within the swept area of the first link. The third version of the W-space planner is unable to deal with situations of this nature. A more complete solution has to be worked out.

Figure 4.29 Collision-free path for the end-effector is not feasible for the robot

3) *Limitation of the fuzzy planner:* It is found in the simulation that different scenario settings require different optimal parameters for the scalar-type fuzzy planner $SF(\left|\vec{pq}\right|)$. Thus a fuzzy planner with adaptive parameter adjustment can be applied.

Chapter 5 Configuration-Space Construction

By the notion of C-space, path planning for a robot with n degree-of-freedom(DOF) can be converted into the problem of planning a path for a point in a configuration space (C-space) of n dimensions. Compared with the path planning executed fully in the W-space, the C-space planning methods can provide *completeness* [7], a property that ensures a path will be found if it does exist.

To ensure that this research work is robust, the focus is now to solve the incompleteness under the framework of W-space planning. A motion planner based on the C-space concept is hence developed to overcome this problem. For this planner design, a practical approach based upon pre-computing the global connectivity in the C-space with respect to all possible obstacle positions in the workspace is proposed. In this proposed approach, the most time-consuming operation is implemented in the offline stage, and online computing only needs to account for the real-time obstacles in the environment; therefore a high computational efficiency for on-line planning is achieved. Furthermore, heuristics including hierarchical C-space representation and multi-resolution searching techniques are applied to speed up the path searching. The developed C-space planner consists of a function for real-time C-space construction and a function for path searching. The former function will be discussed in this chapter, and the latter function will be presented in the next chapter.

5.1 Introduction

In robot motion planning, the calculation of a robot configuration space(C-space) is a very time-consuming task. In particular, when the C-space is represented in a cell-decomposition format, the C-space computation is hindered by the exponential complexity coming from the robot dimensionality [5, 7].

A series of methods have been proposed for the construction of C-space in literature. Some of these methods involve computing the boundary of the obstacles in the C-space(C-obstacle)

analytically. For example, Lozano-Perez considered the case where both the robot and the obstacles are convex polygons or polyhedra, and the C-obstacle boundary for an n -DOF manipulator is approximated by sets of $(n-1)$ -dimensional slices recursively built up from one-dimensional slices [120]. Donald characterized the five dimensional C-space boundaries of a six dimensional C-obstacle using an algebraic format of constraints based on the contact condition [121]. Maciejewski and Fox also studied the analytical description of the boundaries of C-obstacles and derived the connectivity of C-space for revolute manipulators [122]. Zhao *et. al.* developed an analytical representation of C-obstacles using a set of parametric equations [123], which map the boundaries of the obstacles from the W-space into the C-space by pseudo-inverse kinematics. The aforementioned analytical approaches have been demonstrated to be successful only on robots with a small number of DOF. Nevertheless, analytical approaches for C-space construction with large number of DOF under practical dynamic environments have not been studied thoroughly, mainly due to the exponentially increasing complexity in analytically representing the kinematic structures in those applications.

An alternative route is to obtain an approximation of the C-obstacle by using a cell-decomposition method. As an example, a bitmap is widely used to represent C-spaces. A bitmap can explicitly represent the free part of the C-space with 0, and represent the part that gives rise to collisions with an obstacle with 1. Newman and Branicky identified the elemental building blocks that can be easily transformed from the W-space to the C-space [124]. They stored the C-space transforms of shapes as bitmaps first, and then used the superposition of these primitive maps to construct the configuration-space maps for industrial robots. Lozano-Perez and O'Donnell implemented a similar approach to the parallel robot motion planning [82]. These primitive bitmaps, however, are limited to 2-DOF RR type manipulators or Puma-like robots with consecutive parallel rotational joints. Kavraki [125] and Curto and Moreno [126] explored the general method of bitmap calculation for C-obstacle without heuristics, where the C-space maps of obstacles are computed using the Fast Fourier Transform (FFT). This algorithm is based on the observation that the C-space is a convolution of the W-space and the robot. It makes use of the advantages of the

well-established FFT technique including computing hardware implementation. In a different way, Paden computed C-obstacles using a Jacobian-based method [111], where the C-space representation is generated using the uniform bound on the Jacobian.

Unfortunately, no general heuristics exists for the C-space construction, which can be demonstrated by a simple robot A with two DOF given in Figure 5.1. In Figure 5.1(a), point b_1 is located within $|Ob_1| < r_1$ while point b_2 is located within $r_1 < |Ob_2| < r_1 + r_2$, where O is the origin of robot A, and r_1 and r_2 are the length of the links 1 and 2 respectively. The C-obstacles corresponding to b_1 and b_2 are shown in Figure 5.1(b). As it can be seen, the two C-obstacles $CO_A(b_1)$ and $CO_A(b_2)$, which are obtained based on the approach later described in Section 5.2, are quite different.

Figure 5.1 Non-existence of general heuristics for C-space construction

For the approach employing primitive maps in [124], though it is restricted to Puma-like robots with consecutive parallel rotational joints, it provides a design guideline that in cases when relevant portions of the C-space are investigated on the fly, the cost of C-space construction can be reduced greatly. Following this design guideline, a novel two-phase approach is proposed in this chapter for the C-space construction of manipulators. In the first phase, the pre-computation of the C-space connectivity for all possible obstacle positions in

the W-space is performed. The on-line C-space can thus be constructed in the second phase based on the connectivity information obtained. Compared with the primitive C-space bitmap approaches described in [82, 124], the proposed approach is practically applicable to manipulators with any kinematic structure and geometric shape.

It is noticed that on one hand, the C-space representation scheme adopted in this chapter could be constrained by the exponential complexity coming from the robot dimensionality; on the other hand, this representation scheme leads to the feasibility of heuristics like hierarchical tree representation as well as multi-resolution searching strategy, which have shown effectiveness in reducing both the required storage space and the planning complexity [38-40]. Practical industrial applications involve mostly medium and small sized robots with about six DOFs. For such robots, by applying the abovementioned heuristics, the discretized C-space representation is feasible even with the exponential increase of complexity with respect to the dimension, which is demonstrated in this chapter.

5.2 Off-line Phase

The proposed approach to construct the C-space corresponding to a manipulator and a changing environment consists of two phases, namely, an off-line phase and an on-line phase. In the off-line phase, a C-obstacle database (*COD*) for a given robot is developed in which the C-obstacle maps are stored and indexed by the cells of the W-space.

Let $q = (q_1, \dots, q_n)$ denote a configuration of a manipulator with n DOF, where each element of q is a joint parameter, measuring either angular displacement or linear displacement depending on the type of joint.

In the off-line phase, at first, the W-space, denoted as W , is decomposed into cells under certain resolution defined as $W = \bigcup w_i$, with w_i defined as the i -th cell of the W-space. Then, for each cell of the W-space, w_i , we find all configurations of the robot, denoted as Q_{w_i} ,

under which robot A passes cell w_i in the W -space. Mathematically, we define $Q_{w_i} = \{q \mid A_q \cap w_i \neq \emptyset\}$, where A_q refers to robot A in configuration q . Q_{w_i} constitutes the C-obstacle in the C-space corresponding to a point-type obstacle located at position w_i in the W -space, and it is referred to as a C-obstacle map. Given these definitions, all of the C-obstacle maps can be organized into a database, called C-obstacle Database (*COD*), and indexed by w_i .

Here we use a planar manipulator to illustrate the procedure for offline *COD* creation. As shown in Figure 5.2(a), in the x - y plane, there exists an RR manipulator A and a rectangular-type obstacle B . Figure 5.2(b) shows the grid of the W -space W after its decomposition, $W = \bigcup_{i=1}^{n_w} w_i$. As the first step of the proposed approach, the *COD* for robot A will be built in the off-line stage, i.e., for each cell w_i in W , we set up the C-obstacle map Q_{w_i} . Based on the rationale that the robot will collide with an obstacle whenever this obstacle overlaps spatially with any part of the robot, we may develop a procedure for setting up *COD* for robot A shown in Figure 5.2(a). As an illustrative example, for an arbitrary cell w^* selected from W , those configurations constituting Q_{w^*} are given in the W -space (Figure 5.2(c)) and in the C-space (Figure 5.2(d)) respectively. The off-line C-obstacle database can be expressed as

$$COD = \{q \mid A_q \cap W \neq \emptyset\} = \{q \mid A_q \cap (\bigcup w_i) \neq \emptyset\} = \bigcup (\{q \mid A_q \cap w_i \neq \emptyset\}) = \bigcup (Q_{w_i}) \quad (5.1)$$

Figure 5.2 Illustration of the C-obstacle map concept using a planar RR manipulator

Let N_A be the total number of cells of robot A , and let $\theta_{\max}$ and $\theta_{\min}$ be the maximum and minimum value of each joint variable, respectively. We may have the following pseudo code.

Algorithm 5.1 COD setup of a 2-DOF manipulator

Input:

A: a robot with two joints, θ_1 and θ_2

Output:

$\bigcup_{i=1}^{n_w} Q_{w_i}$: set of C-obstacle maps

```

1:   For  $i = 1 : n_w$ 
2:     {  $Q_{w_i} = \emptyset$ ; }
3:   for  $\theta_1 = \theta_{1\min} : \theta_{1\max}$ 
4:     { for  $\theta_2 = \theta_{2\min} : \theta_{2\max}$ 
5:       { Decompose robot A under configuration  $(\theta_1, \theta_2)$ , so that  $A_{(\theta_1, \theta_2)} = \bigcup_{i=1}^{N_A} a_i$ ;
6:         for  $i = 1 : N_A$ 
7:           {  $Q_{a_i} = Q_{a_i} + (\theta_1, \theta_2)$ ; }
8:         }
9:       }

```

5.3 On-line C-space Construction

In the on-line phase, for a real-time scenario containing the same robot A and an obstacle B , the C-obstacle transformed from B relative to A , $CO_A(B)$, can be obtained based on the off-line COD. Indeed, the obstacle B is firstly decomposed under the same grid resolution defined in the off-line phase, *i.e.*, $B = \bigcup b_i$. Then the off-line COD is searched and those maps Q_{b_i} with indexes matching b_i are extracted. According to the union property for C-space [5], the superposition of Q_{b_i} is thus the real-time C-obstacle during manipulator operation, *i.e.*, $CO_A(B) = \bigcup Q_{b_i}$. For the illustrative case shown in Figure 5.2(a), we may assume that the decomposition result of the obstacle B is based on the same resolution used for W , *i.e.*,

$B = \bigcup b_i = \{b_1, b_2, b_3, b_4\}$, as shown in Figure 5.2(e). We then obtain the resultant C-obstacle $CO_A(B)$ as follows:

$$CO_A(B) = \{q \mid A_q \cap B \neq \emptyset\} = \{q \mid A_q \cap (\bigcup_{i=1}^4 b_i) \neq \emptyset\} = \bigcup_{i=1}^4 \{q \mid A_q \cap b_i \neq \emptyset\} = \{Q_{b_1} + Q_{b_2} + Q_{b_3} + Q_{b_4}\} \quad (5.2)$$

Figure 5.2(f) also shows this result of $CO_A(B)$.

As can be seen from the procedure described above, the construction of the *COD* in the off-line phase does not require any knowledge of the obstacles in the real *W*-space. Once the *COD* for a given robot is constructed, it can be utilized as a data source in different application scenarios. Since the time-consuming setup of a *COD* is realized in the off-line phase and only the identification of C-obstacle maps in the *COD* is needed in the on-line phase, the proposed approach is thus expected to be able to deliver real-time result for robot manipulations in dynamic environments. In Section 5.5, we will examine the running time of this proposed approach in some typical cases.

5.4 Representation Scheme

Since the amount of space required for storing the *COD* increases drastically as the dimension of the robot and the resolution in the *W*-space and the *C*-space increases, an efficient representation scheme for an object in the *W*-space and the *C*-space is critical, which is detailed as follows:

5.4.1 Representations of the Robot and the Obstacle in *W*-space

From a geometric point of view, a robot and obstacles can be represented by cells forming their boundaries or by cells forming their entities (referred to as boundary representation and entity representation respectively). We may design four patterns involving different representation modes of a robot, A , and an obstacle, B , in a planar case, as shown in Figure 5.3. In Figure 5.3(a), both A and B are represented by rectangular cells constituting their boundaries; in Figure 5.3(b), A is still represented in boundary, while B is denoted by the cells

wholly covering it; in Figure 5.3(c), A is wholly covered, and B is in boundary; in Figure 5.3(d), both A and B are wholly covered. The C-spaces corresponding to these different geometric representation schemes in the W-space are calculated and the results are shown in Figure 5.3(e) and 5.3(f), from which the following observations are made.

- If robot A is represented by its boundary in the W-space (refer to Figure 5.3(a) and 5.3(b)), the calculated result of superposition $\bigcup Q_{b_i}$ contains the complete boundary and part of internal cells of the C-obstacle $CO_A(B)$ in C-space, as shown in Figure 5.3(e).
- If robot A is represented by the cells wholly covering itself in the W-space (refer to Figure 5.3(c) and 5.3(d)), *i.e.*, the boundary plus all the internal cells, then, no matter how obstacle B is represented in the W-space, the calculated $\bigcup Q_{b_i}$ contains not only the boundary but also all the internal cells of $CO_A(B)$ in C-space, as shown in Figure 5.3(f). This can be explained by the fact that in any configuration where A coincides with B at one of B 's internal cell, *e.g.*, b_{in} , A must at the same time coincide with B at least at one boundary cell of B , *e.g.*, b_{bound} , and therefore $\forall b_{in} \in B, Q_{b_{in}} \subset (\bigcup Q_{b_{bound}})$.

The same observations can be extended to solid-type robots and obstacles.

Based on the above observations and in order to save computational cost as much as possible, the following representation strategies will be used throughout this chapter: if only the boundary of the C-obstacle is sought, we choose boundary representation for both the robot and the obstacle; if the whole entity of the C-obstacle is sought, we use boundary representation for the obstacle and entity representation for the robot.

Figure 5.3 C-obstacle corresponding to different representation mode in W-space

5.4.2 C-space Representation by Non-uniform 2^m -tree

In Chapter 4, a hierarchical data structure called an octree is employed to represent the three dimensional W-space, based upon which the multi-resolution path searching is implemented for the wrist center. Similarly, the C-space of a robot with m DOF can also be represented by a hierarchical tree structure, and such a general hierarchical tree structure with m dimensions is called a 2^m -tree. In a 2^m -tree, each node is an m -dimensional rectangular cell which is labeled as *Black*, *White* or *Grey*, and each *Grey* node has 2^m of children [4]. Actually, an octree is an instance of the 2^m -tree with $m=3$, and the merits of octrees described in Chapter 4, such as space saving and applicability of multi-resolution path searching, also hold true for 2^m -trees. Thus in this chapter, we employ the 2^m -tree data structure to represent a robot C-space. Moreover, we will develop the 2^m -tree in non-uniform format to further reduce the required storage space.

5.4.2.1 Non-uniform 2^m -tree

When the octree was constructed in Chapter 4, the coordinates x , y and z of the W-space are divided into the same number of intervals. If the C-space of a real industrial robot is also divided into the same number of intervals along each joint coordinate, a huge number of cells in the C-space will occur. For example, for a manipulator with six DOF, if a maximum depth of five is assigned to each coordinate, then the total number of cells in the C-space will be $2^{5*6} \approx 10^9$. We may, actually, reduce the size of the required data set.

Firstly, for a manipulator, under the same amount of joint change, the smaller the distance between the joint and the base, the larger the end-effector's maximum movement will be in the W-space. This leads to one category of criterion to determine the joint resolution, that is, the step length along each joint coordinate should result in almost equal displacement of the robot end-effector in the W-space [127, 128]. As shown in Figure 5.4(a), the uniform discretization ($\Delta q_i = \Delta q_j$) results in different Cartesian movements, i.e., $\Delta x_i \neq \Delta x_j$, when different joints i , j are moved. Whereas in Figure 5.4(b), the reasonable discretization

($\Delta q_i \neq \Delta q_j$) results in equal maximum Cartesian movement $\Delta x_i = \Delta x_j$ when different joints i, j are moved. Considering Figure 5.4(b), we may determine the reasonable resolution as

$$\Delta q_i = 2 \arcsin\left(\frac{TipMove}{2l_i}\right) \quad (5.3)$$

where *TipMove* is the required moving accuracy of the end-effector in the W-space, and l_i is the distance between the center of the i -th joint to the tip point of the end-effector. For a spatial robot, this distance depends on the robot configuration. As a conservative estimation, the upper bound of l_i , called $l_{i\max}$, which is the distance between the origin of the i -th joint to the farthest point the end-effect can reach, is adopted. Actually, $l_{i\max}$ corresponds to the l_i value in the worst case. By such a resolution selection, a constraint is added to the movement of the end-effector, which enables the robot to avoid abrupt motions and yield smooth paths.

Figure 5.4 The uniform and reasonable non-uniform discretization (from [127])

Secondly, if the resolution of each joint is sufficiently small, then the swept area between two adjacent link locations is zero, hence no additional collision check has to be made for adjacent link locations and we obtain a reasonable W-space coverage. As shown in Figure 5.5, the maximum joint change satisfying the above requirement is defined as

$$\Delta\phi = \arctan(r/l) \quad (5.4)$$

with r and l the link height and length respectively. Considering a robot manipulator consisting of a kinematics chain, the upper bound of l in the above equation is the length between the i -th joint origin and the robot wrist under the stretched robot configuration. Regarding links of 3D shape, the validity of the above equation also depends on the link geometry. It is valid for cuboid links, and also valid for cylindrical links after adding a security zone around them [128]. We may then have the second category criterion, i.e., choosing appropriate step lengths to obtain a reasonable W-space coverage.

Figure 5.5 Adjacent link positions with swept area zero between them (from[128])

For either criterion, the C-space should be divided into different numbers of intervals along each joint coordinate, which leads to a non-uniform representation of the 2^m -tree. As an example, the depth of a 2^m -tree ($m=5$) to represent the C-space of the robot ESHED SCORBOT ER4pc is obtained according to criteria 1 and 2 respectively, which is shown in Table 5.1. In this example, the 5th joint is an exception whose rotation does not change the position of the end-effector tip, and alternatively a small depth value is manually chosen for this joint. Later in this chapter, we choose the depth value as (5,5,4,3,3) in simulation studies based on the first criterion.

To illustrate the non-uniform 2^m -tree, Figure 5.6 shows a non-uniform quadtree example (where $m=2$). As can be seen from Figure 5.6(c), each node of the quadtree is equally divided into 4 children at both depth 1 and 2; but at depth 3, the map is divided only along the latitudinal coordinate, i.e., each node is divided into 2 sub-nodes only. The concept of *Morton code* and *directional digit* can also be applied here. As shown in Figure 5.6(c), the black node with Morton code 211 is tracked, starting from the root and traversing along the dotted line, following the corresponding directional digit at each depth. Alternatively, the sample space in

Table 5.1 Process to determine the resolution leading to reasonable W-space coverage

Joint		1	2	3	4	5
r(mm)		150	70	70	40	15
l(mm)		595	580	360	140	0
Cat. 1	resolution ratio (... $\arcsin(\frac{1}{2l_i})$: ...)	$\frac{1}{595}$	$\frac{1}{580}$	$\frac{1}{360}$	$\frac{1}{140}$	NA
	rounded resolution ratio	$\frac{1}{2^5}$	$\frac{1}{2^5}$	$\frac{1}{2^4}$	$\frac{1}{2^3}$	$\frac{1}{2^3}$
Cat. 2	$\Delta\phi_i = \arctan(r/l)$	14.3°	7°	11°	16.4°	NA

Figure 5.6(a) can be divided along only the latitudinal coordinate at depth 1 and be divided along both coordinates at both depth 2 and 3, which results in a different but equivalent non-uniform quadtree, as shown in Figure 5.7. In the rest of this thesis, the first convention to decompose the 2^m -tree non-uniformly will be employed.

(a) A 2D sample space

1				310	311	33
				300	301	32
010	011	030	031	210	211	23
000	001	020	021	20	22	

(b) Non-uniform quadtree decomposition and node labeling

(c) Quadtree representation of the sample space

Figure 5.6 Quad tree demo for non-uniform 2^m -tree

01		03		111	113	12	
				110	112		
001	003	021	023	101	103	121	123
000	002	020	022	100	102	120	122

(a) Another format of non-uniform decomposition

(b) Quadtree demo under another decomposition convention

Figure 5.7 Non-uniform space decomposition under another convention

5.4.2.2 Encoding the C-obstacle Maps

As described earlier, a single C-obstacle map in a *COD* denotes the C-space with respect to an obstacle particle in the W-space and is represented by a hierarchical non-uniform 2^m -tree. Several typical techniques are available to implement such a non-uniform 2^m -tree in programming.

a. Pointer Structure

It is convenient to represent a 2^m -tree using pointers, by allocating 2^m pointers per node, each of which points to one of its children, like in [116]. However, explicitly representing a 2^m -tree by pointers is prohibitively expensive in terms of memory requirements [113].

b. Linear 2^m -tree

This method is a general extension of the linear octree mentioned in Section 4.2.2. It assigns a unique key to each black leaf node, which encodes both the location and the size of the cell [117]. The locational part can be denoted by the *Morton code*, and the size information is

denoted by the depth value. Compared with a pointer-type data structure, the linear 2^m -tree is much more space saving.

c. Concise 2^m -tree with Bit Encoding

In Figure 5.8, two arrays with bit encoding are adopted to represent the quadtree originally appeared in Figure 5.6. In the Black/white array, a single binary bit is used to denote whether the corresponding tree node contains *Black* element, *i.e.*, a bit value 0 represents that the node is *White*, and a bit value 1 means the node is *Black* or *Grey*. Similarly, in the Leaf array, a bit value 1 represents a non-leaf node and a bit value 0 refers to a leaf node. A similar representation is used in [129] for thinning of three dimensional objects.

Depth	Black/white	Leaf
0	1	1
1	1011	1011
2	1111, 0101, 1100	1111, 0100, 1100
3	01, 01, 10, 10, 01, 01, 01	00, 00, 00, 00, 00, 00, 00

Figure 5.8 Representing the quadtree in Figure 5.6(c) by scheme c

The space requirements of the above techniques are summarized in Table 5.2. As can be observed from this table, the scheme *b* and *c* need comparative space to store the example quadtree, which is much less than scheme *a*. The main difference between scheme *b* and *c* is that scheme *c* stores not only the black nodes but also the grey nodes. As described later in Chapter 6, this property of scheme *c* eliminates the need of re-planning when multi-resolution path searching is applied. Thus representation scheme *c* is chosen in this research.

It is worth noting that certain trade-off exists for the scheme *c* of 2^m -tree representation. Before any operation about the 2^m -tree proceeds, the data stored in the tree representation needs to be decoded from the compressed bit encodings, which could to certain extent slow down the speed for tree traversing and other operations such as union and subtraction.

Table 5.2 Comparison of three representation schemes

(Here n_g and n_b is the number of grey nodes and black nodes in the tree separately, h is the maximum depth of the tree, and m is the degree of the tree)

Scheme	Theoretical Space requirement	Actual space to represent the quadtree in Figure 5.6(c) (Bytes)
Pointer structure	Space $> 4 * n_g * 2^m$ (normally a pointer needs 4 bytes of memory)	> 160
Linear 2^m -tree	Space $= n_b * (m * h + \log(h+1)) / 8$, [115]	8
Concise 2^m -tree with bit encoding	$n_b < \text{Space} < 2^m * n_b$	$62/8 \approx 8$

5.4.2.3 Decoding and Superimposing C-obstacle Maps

At the on-line stage, those maps recognized from the *COD* are superimposed to construct the required real-time C-space. Let $L_{\max}$ be the preset depth bound, N_b be the total number of obstacle cells respectively. We may have Algorithm 5.2 in pseudo code.

Algorithm 5.2 Superposition of C-obstacle maps

Input:

$\bigcup_{i=1}^{N_{map}} Q_i$: robot *COD* consisting of C-obstacle maps

$\bigcup_{i=1}^{N_b} b_i$: obstacle distribution

Output:

A_{tree} : 2^m -tree representing the current environment

```

1:   Define a blank array  $A_{tree}$  to denote the fully extended  $2^m$ -tree
2:   for  $l=1:L_{\max}$            //  $L_{\max}$  is the maximum depth of the  $2^m$ -tree
3:       for  $k=1:2^{l*m-3}$  //the  $l$ -th level contains a total of  $2^{l*m}$  nodes, represented in
            $2^{l*m} / 2^3$  bytes
4:           {  $A_{tree}(l, k) = 0$  // initialization }
5:       for  $l=1:L_{\max}$ 
6:           { for  $n=1:N_b$ 
7:               { for  $k=1:2^{l*m-3}$ 
8:                    $A_{tree}(l, k) = A_{tree}(l, k) | Q_n(l, k) *$  //scan sequentially the  $n$ -th
//C-obstacle map  $Q_n$  and fill the grey and black nodes into the  $l$ -th row
9:               }
10:          }
```

* | refers to the bitwise logic “or” operation

The resultant A_{tree} is the C-space representation corresponding to the current environment. During the scanning process, the location of a grey node cell C , which is located at depth l , is calculated as in Equation 5.5.

$$Location(l, C) = Local_offset + 2^m * Location(l-1, parent(C)) \quad (5.5)$$

where the local offset value, $Local_offset$, is between 0 and 2^m , $parent(C)$ denotes the parent node of C . As an example, Figure 5.9 gives the result after the condensed quadtree in Figure 5.8 is scanned and filled into A_{tree} .

Depth	Black/white	Leaf
0	1	1
1	1011	1011
2	1111,0000,0101,1100	1111,0000,0100,1100
3	01011010,00000000,00010000,01010000	00000000,00000000,00000000,00000000

Figure 5.9 The complete 2^m -tree after bit decoding

5.4.3 Techniques to Reduce the Space Consumption of COD

5.4.3.1 Optimal Resolution Selection

For the proposed approach of C-space construction, the selection of resolution in the W-space and the C-space have a significant impact on both the computational cost and the accuracy. Several sets of resolutions are tested for the aforementioned illustrative case shown in Figure 5.3. The calculated C-obstacles are shown in Figure 5.3 and Figure 5.10, which lead to the following observations.

- Under a fixed resolution in the W-space, as the resolution in the C-space gets higher, the calculated C-obstacle boundary gets smoother and converges continuously to the theoretical C-obstacle boundary, as illustrated in Figure 5.10(a), Figure 5.10(b), and Figure 5.10(c).
- Under a fixed resolution in the C-space, the increase of resolution in the W-space leads to the same trend, which is reflected in Figure 5.10(d), Figure 5.3(c), and Figure 5.10(e).

Normally, the precision requirement is determined by a real robot application, hence under the premise that the accuracy is met, low resolution can be chosen to reduce the computational cost.

5.4.3.2 Reference C-obstacle Map

As an approximation, the C-obstacle map corresponding to an arbitrary cell in the W-space may be represented by the bitmap corresponding to the nearest neighbouring reference cell in the W-space, then the total number of C-obstacle maps in a *COD* can be reduced to the number of reference cells in the W-space, hence the *COD* size can be reduced. The neighbourhoods of cells in the W-space can be obtained by tree decomposition. The appropriate density of reference cells in the W-space depends on the kinematic structure and geometric dimensions of the robot. Figure 5.11 shows an approximation of a C-obstacle map corresponding to the case given in Figure 5.3(c). Under the adopted resolution, the deviation between the approximate C-obstacle and the original one is noticeable but small.

5.4.3.3 Interpolation of the Reference C-obstacle Maps

Another reasonable approximation of the C-obstacle map corresponding to an arbitrary cell in the W-space can be made by using the linear interpolation between the reference C-obstacle maps. The principle of this linear interpolation is illustrated in Figure 5.12, where a sample point in the W-space and its four neighbouring reference points are given in Figure 5.12(a), and Figure 5.12(b) presents a two-step interpolation process with weights allocation to obtain the C-space corresponding to the sample point. Such a scheme is analogous to the morphing procedure in image processing that employs all nodes as the so-called key points.

Figure 5.10 Calculated C-obstacle under different resolutions

(In this figure and in the rest of this chapter, the solid lines in the C-spaces denote the theoretical C-obstacle boundaries, which are obtained from analytic derivation)

Resolution: 128x128 in W, 128x128 in C

Density of reference cells in W: 32x32

Figure 5.11 C-space calculated based on neighbour encoding

(a) Reference grid in W-space

(b) Obtain CO_p by linear interpolation (with allocated weights shown on the arrow)

Resolution: 128x128 in W, 128x128 in C

Density of reference cells in W: 32x32

(c) Calculated example C-obstacle map through interpolation

Figure 5.12 Demonstration of C-obstacle interpolation in a 2D case

By this interpolation technique, the task of on-line C-space construction is now transformed into the superposition of reference C-obstacle maps, which is summarized below:

$$CO_W = CO_{\cup w_i} = \cup CO_{w_i} = \cup (CO_{w_{xi1}} \oplus CO_{w_{xi2}}) = \cup ((CO_a \oplus CO_b) \oplus (CO_c \oplus CO_d)) \quad (5.6)$$

here $\oplus$ denotes the interpolation between two bitmaps. Figure 5.12(c) shows the C-obstacle calculated by using this linear interpolation that also corresponds to the case given in Figure 5.3(c). Compared with the result by pure neighbourhood encoding, the result calculated by linear interpolation matches the original boundary better. The interpolation can also be expressed in a one-step manner as shown in Figure 5.13.

Figure 5.13 Another interpolation approach

5.4.3.4 Comments on Reference C-obstacle Maps and Their Interpolation

- Both reference methods make use of neighbourhood in the W-space and can reduce the *COD* storage space.

The result from either method is sensitive to the resolution. The interval allowed between the uniform reference points in the W-space is determined by the robot kinematics structure and dimensions of the links. It can be deduced that when the reference cells in the W-space distribute too sparsely, the C-space obtained will depart from the true value too much to suit motion planning.

5.5 Simulation Results

In this section we perform case studies to verify and demonstrate the proposed approach. Some of the simulation scenarios have been presented in the previous sections as comparative examples. Here we focus on the computational efficiency and complexity analysis.

5.5.1 C-obstacle Construction for RRR Robot

In this case, a three DOF spatial manipulator is adopted as the prototype. Its three links are modelled as proximate cylinders with the same dimension (radius=25mm , length=100mm). Each joint is allowed to move from 0 to 360°. A cuboid obstacle is placed within the reachable range of the robot.

The W-space scenario and the calculated C-obstacles are shown in Figure 5.14, and the simulation settings and computation performance are summarized in Table 5.3. In the simulation, the neighbourhood of W-space cells is utilized and the density of reference cells is listed in Table 5.3. In [82], the time to build a $64 \times 64 \times 64$ configuration space for the first three DOF of the Puma robot on a ‘Thinking Machines’ Connection Machine with 8k processors is approximately 2 seconds. Compared with the result above, the on-line running time of the proposed approach is faster if the two computing platforms are deemed comparative.

Figure 5.14 Simulation results of a spatial RRR manipulator**Table 5.3 Settings and calculation result of the 3-DOF robot**

Settings in W-space	Range of space	$[-250, 250] \times [-250, 250] \times [-250, 250]$ (mm,mm)
	Resolution	$3.9mm, 3.9mm, 3.9mm$
	No. of cells	$128 \times 128 \times 128 = 2.1 \times 10^6$
	No. of reference cells	$64 \times 64 \times 64 = 2.62 \times 10^5$
Settings in C-space	Range of space	$[0, 360^\circ] \times [0, 360^\circ] \times [0, 360^\circ]$
	Resolution	$2.81^\circ, 2.81^\circ, 2.81^\circ$
	No. of cells	$128 \times 128 \times 128 = 2.1 \times 10^6$
Storage scheme		2^m tree + bit encoding + neighbourhood encoding
Running time	COD setup	≈ 2 hours
	C-space construction for scenario in Figure 5.14(a)	0.26(sec.)
Storage space		24MB

5.5.2 C-space Construction for a 5-DOF Robot

Currently, manipulators with five or more DOF are widely used in industry. Clearly, constructing the explicit C-space for these types of manipulators is quite useful for practical applications. It is worth emphasizing that due to the system complexity in kinematic structure and geometric shape, very few successful studies on C-space construction for such

manipulators have been reported in the literature. In this simulation case, the ESHED SCORBORT ER4pc Robot is employed as the prototype to evaluate the proposed approach in a very complex scenario with a narrow passage, which is latter used to test the developed C-space motion planner in Chapter 6.

Off-line COD calculation based on the robot CAD models

This step only involves the robot CAD model. First, the robot geometry, including its end-effector (EE), is modeled in Pro/E software and then saved in format of .obj files. These .obj files store the triangles constituting the surfaces of robot links. In the off-line phase, these .obj files are read by the program and the manipulator is re-assembled in the simulation scenario according to the robot kinematics. Denoting the robot as A , at configuration q , the robot links are decomposed into the collection of cuboid cells, denoted as $A(q) = \sum a_i$. Figure 5.15 shows such a decomposition result. Then, the configuration q will be added to those C-obstacle maps indexed by a_i as $Q_{a_i} \leftarrow Q_{a_i} + q$. The above procedure repeats until q traverses the C-space under a preset resolution and then the COD of this whole robot is obtained.

Figure 5.15 Decomposition result of the SCORBOT robot at one configuration during off-line COD creation

On-line C-space construction

Based upon the obtained *COD*, the C-space corresponding to the environment will be constructed. As shown in Figure 5.16, in this scenario a wall is arranged so that a narrow passage is left for the robot.

Figure 5.16 ESHED SCORBOT-ER4pc Robot and the obstacle

Following the same resolution to decompose the robot, the wall is also decomposed into cells, expressed as $B = \sum b_i$, and Figure 5.17 presents this decomposition result. The required C-space is then assembled by superposing the C-obstacle maps as $CO_A(B) = \bigcup Q_{b_i}$.

The simulation parameters and the computational performance are listed in Table 5.4. While the off-line running time based on the proposed approach is comparable to the results presented in [128], the on-line running time is only about 2.0 seconds. It is also noted this on-line running time corresponds to the superposition of C-obstacle maps under the maximum depth. As will be verified in Chapter 6, under the multi-resolution strategy, the developed C-space planner based upon the proposed method for C-space construction can find a collision-free path in the same scenario in much shorter time.

Figure 5.17 The geometric decomposition result of obstacles

Table 5.4 Settings and calculation result employing the SCORBOT Robot

Settings in W-space	Range of space(mm)	$[-600, 600] \times [-600, 600] \times [0, 1200]$
	Resolution(mm)	18.75, 18.75, 15.625
	Number of cells	$64 \times 64 \times 64 = 2.62 \times 10^5$
	Number of obstacle cells after the wall is decomposed	1840
Settings in C-space	Range of C-space($^{\circ}$)	$[-130.0, 180] \times [-125.0, 40.0] \times [-130.0, 130.0]$ $\times [-220.0, 40.0] \times [0, 360]$
	Resolution of joints($^{\circ}$)	4.84, 5.15, 16.25, 32.5, 45.0
	Total Number of cells	2×10^6
Storage scheme		2^m -tree + bit encoding
Running time	COD of-line setup	≈ 4 hours
	C-space construction for the scenario in Figure 5.16	2.015sec.
Storage space		110MB

5.6 Complexity Analysis

The worst case time complexity of the proposed algorithm can be analyzed as follows:

- In the off-line phase, the complexity of creating a *COD* is $O(C_W \cdot C_C)$, where C_W denotes the complexity of the robot W-space and it can be expressed as $\prod_{d=x,y,z} (N_d)$ with N_d the resolution in each dimension of the W-space respectively; C_C denotes the complexity of a C-obstacle map and it can be expressed as $\prod_{d=1}^n (N_d)$ with N_d the resolution in each dimension of the C-space, and n is the number of DOF. Therefore, the complexity of the *COD* can be denoted as $\prod_{d=x,y,z,1,\dots,n} (N_d)$.
- In the on-line phase, the complexity for superposing the C-obstacle maps can be expressed as $O(C_o \cdot C_C)$, where C_o is the complexity of the real obstacles in the W-space, C_C is still the complexity of a C-obstacle map.

In typical robot applications, the obstacles occupy only a small part of the robot W-space, thus the complexity of obstacles C_o is far lower than that of the W-space C_W . Furthermore, the hierarchical 2^m -tree representation scheme of the C-obstacle maps can greatly reduce the time complexity in the on-line phase. As an example, in the aforementioned case study 2, the off-line phase of *COD* creation costs about 4 hours while the on-line C-space construction corresponding to a wall only needs 2.015sec.

It should be noted that with the further increase of the number of DOF, the storage space requirement could increase drastically. For those applications, the inherent representation of the C-space connectivity, such as the dynamic roadmap [37] could be a better choice.

5.7 Conclusion

In this chapter, a new two-phase approach for C-space construction of manipulators is proposed. In the off-line phase, a C-obstacle database (*COD*) which stores the C-obstacle

maps indexed by the cells of the W-space is created. Based on the off-line *COD*, one can obtain the C-space under real-time operation. The proposed approach is general in nature and its on-line running time is extremely short. Simulations on 3-DOF and 5-DOF manipulators have verified the improved performance, and suggested the potential for real-time applications under dynamic operating conditions.

Chapter 6 Path Searching in Configuration Space

In Chapter 5, the *COD* is pre-computed for a given manipulator. Once established, it can be utilized later in any motion-planning applications involving the same manipulator. In each cycle of on-line motion planning, the C-space corresponding to the current environment can be obtained by superposing those C-obstacle maps whose indices match the real obstacle cells in the environment. To find out a collision-free path in the constructed C-space effectively, an efficient searching algorithm is required to fully utilize the hierarchy of the non-uniform 2^m -tree. Moreover, to find out all of the free neighbours for an arbitrary free cell in the tree is a prerequisite for path searching. In this chapter, these two issues will be addressed.

6.1 Neighbour Searching in a Non-Uniform 2^m -tree

As described in Chapter 5, after superposition, the C-space corresponding to the current obstacle distribution in the environment is represented in a non-uniform 2^m -tree format. Ultimately a collision-free path will consist of a list of neighbouring free cells in this 2^m -tree. Therefore, the neighbour searching in such a 2^m -tree plays a fundamental role for path finding. In chapter 4, a neighbour-searching method in top-down fashion is implemented by bit operations in an octree ($m=3$); whereas in this chapter, the planning space being studied has about six DOF; thus, the higher dimension and the non-uniformity of the adopted 2^m -tree representation makes the task of neighbour searching more complex than that in an octree.

As mentioned earlier, in an octree, the neighbours of a cell may be located in six directions, i.e., x_+ , x_- , y_+ , y_- , z_+ , z_- . When extended to a general non-uniform 2^m -tree, the neighbours with respect to a single cell may be located in totally 2^m directions. As an example, the planar sample corresponding to a quadtree ($m=2$) in Figure 5.6 is considered as the illustrative object. In this example, the directions for possible free neighbours with respect to a cell whose Morton code is 20 are checked, and the result is shown in Figure 6.1. In this case, there is an exceptional direction that when $y_- = -1$, the potential neighbour will be located

outside of the space, thus this direction is invalid for neighbour searching.

Figure 6.1 Sample space demonstrating the directions of free neighbours

Table 6.1 lists the bit operations to obtain all these directions, with the meanings described for each column similar to those in Table 4.4.

Table 6.1 Procedures to find all directions for free neighbour searching

Reference cell			Free neighbours			
Morton code	Bit value	Bit value along coordinate	Possible bit value of neighbours	$\oplus$ Operation	Bit value	Morton code
20	10 00	x=1 0	$x_- = 0 \ 1^*$	$x_- \oplus y^{**}$	00 10	02
			$x_+ = 1 \ 1$	$x_+ \oplus y$	10 10	22
		y=0 0	$y_- = -1 \ 1$			
			$y_+ = 0 \ 1$	$x \oplus y_+$	10 01	21

* With respect to cell 20, $x_- = x-1 = 10 - 1 = 01$. The same meaning applies to x_+ , y_+ , y_-

** $\oplus$ denotes the operation of bit interleaving

At each possible direction, the neighbour cells in the quadtree are examined whether they are free from depth 1 till depth $d_{\max}$, with $d_{\max}$ being the maximum depth of the quadtree. The result of the above neighbour searching process is shown in Figure 6.2, where each dashed line refers to a searching thread. When a searching thread terminates at a white leaf cell, it means a valid free neighbour is found. The procedures to find all these free neighbours are explained in detail in what follows.

Figure 6.2 A non-uniform quadtree sample demonstrating the free-neighbour searching

The first searching direction has Morton code 02, and the directional digit at depth 1 is 0. In Figure 6.2, Cell 0 at depth 1 is grey; thus, we go downwards to check depth 2. At depth 2, Cell 02 is found as grey; thus, we continue to go downwards to depth 3. Note that at depth 3, since the space is divided only along the x coordinate, only two children of Cell 02 exists at this depth, i.e. cell 020 and 021. Among these two children, Cell 021 shares its boundary with Cell 20 while Cell 020 does not. This can be deduced from the fact that the current searching direction is x_- , which means only those child cells with a bit value 1 along the x coordinate are neighbours of Cell 20. Again from Figure 6.2, Cell 021 is found as white, thus we add it into the free neighbour list. Since the maximum depth of the quadtree is already reached, the neighbour search at this direction is completed.

Now consider the direction of 22. At depth 1, Cell 2 is grey, thus we go downwards to depth 2;

at depth 2, Cell 22 is found to be white, thus we add Cell 22 into the free-neighbour list of Cell 20.

At last, we consider the direction of 21. At depth 1, Cell 2 is grey, so we go downwards to depth 2; at depth 2, Cell 21 is also grey, hence we further go down to depth 3; again, only two children of Cell 21, i.e., 210 and 211, exist at depth 3. Since the space is divided only along the x coordinate at depth 3, and the current searching direction is y_+ , Cell 210 and 211 are both neighbours of Cell 20. From Figure 6.2, Cell 210 is white but cell 211 is black. Thus, we add 210 into the free-neighbour list and the searching in this direction is completed.

It is also noted that during the above process for neighbour searching, we visited depth 3 of the quadtree twice. At the first time, Cell 021 and 020 are checked, among which Cell 021 is a neighbour of Cell 20 whereas Cell 020 is not; at the second time, Cell 210 and 211 are checked and both of them are neighbours of Cell 20. This is caused by the non-uniformity of this quadtree, which is verified from the original space representation shown in Figure 6.1(a).

The above process of neighbour searching can be summarized by the following Algorithm.

6.2 On-line Path Searching under Multi-resolution Strategy

Since the A* algorithm is most suitable for cases when the shortest path is desired [35], it is employed here to search for collision-free paths in an m -dimensional C-space. The following evaluation function is adopted for path scoring:

$F = G + H$, where

- G = the movement cost to move from the starting point A to a given cell in the C-space, following the path generated to get there.
- H = the estimated movement cost to move from that given cell in the C-space to the final destination, point B. The length of a straightforward line connecting these two positions and ignoring obstacles serves as the heuristic guess value of H .

Algorithm 6.1 Free neighbour search in a non-uniform 2^m -tree**Input**C: free reference cell with Morton code M_C **Output**

FN(C): free-neighbour list of cell C

```

1:  For  $i=1:2^m$  //a total of  $2^m$  directions for free neighbour searching
2:  {    $L=1$ ;
3:      While ( $L \leq L_c$ ) //  $L_c$  is the depth of the reference cell
4:      {   Extract Morton code  $M_{il}$  by bit interception at depth  $L$ ;
5:          If (cell  $M_{il}$  is White)  $FN(C) \leftarrow FN(C) \cup M_{il}$ ;
6:          If (cell  $M_{il}$  is Black) return;
7:          If (cell  $M_{il}$  is Grey)  $L=L+1$ ;
8:      }
9:       $L=L_c+1$ ;
10: While ( $L \leq d_{\max}$ ) //  $d_{\max}$  is the maximum depth of the  $2^m$ -tree
11: {   //At depth  $L$ , half number of children or all of them
12:     //share face with the reference cell;
13:     If (all children are neighbours)  $Num=2^m$ ;
14:     else  $Num=2^{m-1}$ ;
15:     For  $j=1: Num$ ,
16:     {   Obtain Morton code  $M_{ilj}$  by bit shift and addition;
17:         If (cell  $M_{ilj}$  is White)  $FN(C) \leftarrow FN(C) \cup M_{ilj}$ ;
18:         If (cell  $M_{ilj}$  is Black), continue;
19:         If (cell  $M_{ilj}$  is Grey),  $L=L+1$ ;
20:     }
21: }
22: }
```

As verified in Chapter 4, the multi-resolution searching strategy can greatly improve the efficiency of path searching when the planning space is represented by a hierarchical tree structure. In this chapter, on-line paths need to be found in an m -dimensional C-space, with m about six for typical industrial robots. Thus, higher dimensionality needs to be taken into account when applying the multi-resolution strategy.

The proposed on-line path planning process is outlined in Figure 6.3, where the operations of C-space construction and path searching are executed level by level. First, the obstacles in the workspace at the current moment are decomposed into cells. Starting from depth 1 (the lowest

resolution), the C-space representation with the current resolution is obtained by map superposition. The C-space is then searched by A* algorithm, which attempts to find a collision-free path. If no path is available, the algorithm switches to the next depth and repeats the above procedure using a higher resolution. This planning sequence is repeated until a global collision-free path connecting the current robot configuration to the goal, noted as $\{Path(t_i, \lambda) \mid Path(t_i, 0) = q(t_i), Path(t_i, 1) = q_{goal}\}$, is obtained or the highest resolution is reached and no collision-free path is found. At the end of this planning cycle, the robot moves a step length in the C-space along $Path(t_i, \lambda)$. The above procedure repeats until the robot reaches the goal. Under this multi-resolution planning strategy, a path can be found at a lower resolution, which reduces the on-line computational time. Furthermore, the above searching process is executed only within free cells, which avoids re-planning.

Algorithm 6.2 On-line Path Planning in the C-space

Input: q_{init} : the init configuration q_{goal} : the goal configuration**Output:** $Path(t, \lambda)$: a motion sequence of robot and $Path(t_0, 0) = q_{init}, Path(t_{end}, 1) = q_{goal}$

```

1:    $i \leftarrow 0, t \leftarrow t_0$ 
2:   Repeat
3:     Update environment,  $B \leftarrow B(t_i)$ ;
4:     decompose  $B(t_i)$  as  $B(t_i) = \bigcup b_i$ ;
5:      $L \leftarrow 0$ 
6:     Repeat
7:       calculate C-space as  $CO_A(B) = \bigcup(Q_{b_i})$ ;
8:       Search path in the C-space by calling  $A^*(q, q_{goal}, CO_A(B))$ ;
9:        $L \leftarrow L + 1$ ;
10:    Until a collision-free path is found:  $Path(t_i, \lambda) = A^*(q, q_{goal}, CO_A(B))$ ,
    and  $Path(t_i, \lambda)|_{\lambda=0} = q, Path(t_i, \lambda)|_{\lambda=1} = q_{goal}$ ;
11:    Move robot with a step length:  $q_i \leftarrow Path(t_i, \Delta)$ ; //  $\Delta$  is a small constant
    //denoting the step length along the path
12:     $i \leftarrow i + 1, t \leftarrow t_i + T$ 
13:  Until  $q = q_{goal}, i_g \leftarrow i$ 
14:  Return  $\sum_{j=0}^{i_g} q_j$ 

```

Figure 6.3 Flowchart of multi-resolution path planning in the C-space

An illustration is performed to verify the proposed multi-resolution strategy, as shown in Figure 6.4. In this example, a robot prototype which takes the first three DOFs of the ESHED SCORBOT ER4pc robot is adopted. Figure 6.4(a) and Figure 6.4(b) are the starting and the goal configurations respectively. Using the multi-resolution strategy, the A* algorithm terminates at depth=3 and finds the path as shown in Figure 6.4(c), which is a list of consecutive free cubes in the C-space. As can be observed from the result, for this simple case, the path can be found at a coarse resolution in the C-space. During this path searching process, those cells visited by the planner (the cells constituting the C-obstacle and the path in Figure 6.4(c)) only occupy a quite small portion of the entire C-space, which shows the improved

on-line efficiency due to the multi-resolution strategy.

(a) Start configuration

(b) Goal configuration

(c) The path found in C-space

Figure 6.4 Sample case verifying the A* algorithm under multi-resolution strategy

In order to increase the path searching efficiency, the following techniques are employed in the proposed approach.

6.2.1 Maintaining the OPEN List by Binary Heaps

Maintaining the OPEN List is actually one of the most time-consuming steps in the A* searching. At each step of the on-line searching, we only require that the cell with the lowest

evaluation cost can be easily accessible at the top of the list. A data structure named binary heap [130, 131] meets this requirement and is employed in this research. A binary heap is a collection of items in which either the lowest or highest value item is at the top of the heap. It can be viewed as a binary tree with two additional constraints:

- 1) the shape property, i.e., the tree is either a perfectly balanced binary tree (all leaves are at the same level), or, if the last level of the tree is not complete, the cells are filled from left to right;
- 2) the heap property, i.e., each cell is greater than or equal to each of its children according to some comparison predicate which is fixed for the entire data structure [130].

A binary heap can be saved in a simple, one dimensional array [131].

6.2.2 Lazy Evaluation of Neighbours

To facilitate the path searching in a C-space, a connectivity graph which represents the neighbourhood between all the free cells in the equivalent 2^m -tree can be constructed. However, if the aforementioned neighbour searching method is directly applied in calculating the adjacency between the permutations of all free cells in the tree, the computation complexity increases exponentially as the depth of the 2^m -tree increases. Our simulation verifies that it takes tens of seconds to construct such a connectivity graph with depth=5 for the robot prototype in Figure 6.4, which is not feasible for on-line motion planning. Alternatively, a lazy strategy for neighbour searching is adopted in this research in order to reduce the time required for the search. The lazy strategy has been previously utilized to speed up the construction of probabilistic roadmap [45, 132], where the cells and edges along the path are checked for collision until the query phase. Similarly in this research, the free neighbours of a free cell are searched only when this free cell is currently the interested cell during the path searching, i.e. the first cell in the OPEN list. This lazy strategy for neighbour searching has provided good efficiency in our planner implementation. For the same robot application in Figure 6.4, the time to find the valid path shown in Figure 6.4(c) (including the time for C-space construction and the time for A* path searching) is reduced to less than 10 milliseconds during the simulation.

6.2.3 Self-collision

In this chapter, all those configurations that lead to robot self-collision are recognized and stored into a single C-obstacle map Q_{self} . In the on-line stage, the C-space obtained by map superposition is further superposed with the single map Q_{self} containing all self-collision configurations. Thus those configurations involving self-collision are recognized as forbidden areas in the 2^m -tree.

6.2.4 Path Generation and Smoothing

Applying the A* searching algorithm to a 2^m -tree leads to a list of free cells which define a path in the C-space between the starting and goal cells. One simple way to obtain the executable path for the robot is just connecting the center of the consecutive blocks, which is adopted later in this chapter during simulations. Since the C-space is represented by a non-uniform 2^m -tree, the free cell size in the tree might be different. Thus, it may occur that the line connecting two free neighbours might penetrate through additional non-free cells, which will result in an unfeasible path. Figure 5.6(b) shows an example, where a line connecting cells 1 and 031 interferes with black cell 030. If the connecting line between two free neighbours is constrained to pass through their common face, the generated path can be guaranteed as safe.

To make the obtained path smooth enough for the robot to execute, the B-spline curve has been applied. The advantages are that the change to a control point only affects the curve in that locality, and an arbitrary number of points can be added without increasing the degree of the polynomial. Among the B-spline curves, the cubic B-spline (with order 3) is most often used due to its positional, first-derivative and second-derivative continuities [133]. In this chapter, the cubic B-spline is applied to smooth the rough path in the C-space, which is generated by connecting the free cell centers into smooth path with second-derivative continuity.

6.3 Simulation

The flowchart of the developed C-space motion planner is given in Figure 6.3, where two loops are included. The inner loop denotes the planning actions within an on-line planning cycle, where the C-space construction and A* path searching is executed recursively, until a collision-free path, $Path(t_i, \lambda)$, connecting the current robot configuration to the destination is found. At the end of this planning cycle, the robot moves a step length along $Path(t_i, \lambda)$. At the outer loop, the planning cycle is executed recursively until the robot reaches its destination. In this section, different simulation cases have been executed to verify and demonstrate the proposed motion-planning approach.

6.3.1 Case 1 - Simulation in Static Environment

In the first simulation case, the ESHED SCORBOT ER4pc robot with five DOF is adopted as the prototype. A static scenario named “Bottleneck”, as shown in Figure 6.5, is adopted to test the planner’s ability in traversing a narrow passage. The same scenario has been applied in Section 5.5.2 to test the time for on-line C-space construction and 2.015 seconds is reported. In this scenario, the line connecting the start and the goal positions of the end-effector hits the wall. Therefore, to traverse through it, the robot arm should climb up to the height of the narrow passage and then pass through it. Fine discretization in both the W-space and the C-space is needed to facilitate the path searching near the narrow passage. In terms of geometric complexity, the surface model of the SCORBOT robot prototype consists of a total of 1544 triangles, and the wall is decomposed into a total of 1840 obstacle cells.

In this simulation case, only a static obstacle is involved, thus only one planning cycle is executed. By employing the developed planner, a collision-free path is found when the searching reaches depth=4 in the C-space, and the corresponding trajectory of the end-effector is shown in Figure 6.5(c). The total on-line planning time is 0.882 second, among which the time for A* path searching is 0.010 second, which is very short as compared with the time for

map superposition of 0.681 second. Overall, the planner found the path through this narrow passage in acceptable time duration under the adopted resolution.

(a) Initial configuration

(b) Goal configuration

(c) One screenshot from simulation case 1

COD database settings

W-space: Number of cells: 64x64x64

Resolution: 18.75, 18.75, 15.625 mm

C-space: Number of cells: 32x32x16x8x8=10⁶

Resolution: 4.84°, 5.15°, 16.25°, 32.5°, 45.0°

COD Size: 110 MByte

Figure 6.5 Static simulation scenario

Recalling that in Section 5.5.2, the time for C-space construction of the same scenario is executed until the maximum tree depth ($=5$) is reached, while here the path searching returns a collision-free path at depth= 4 with a time duration less than half of the former. This clearly verifies the efficiency of the multi-resolution strategy.

6.3.2 Case 2 - Dynamic Wall

In the second simulation case, a simple dynamic scenario is designed to test the planner's performance. As shown in Figure 6.6, a rectangular wall is located between the initial and goal configurations, and it keeps moving upward, thus the robot should adapt to its motion to the moving wall accordingly. The same *COD* settings as that used in Section 6.3.1 are adopted. The simulation result is given in Figure 6.6. The robot initially plans to go around the wall from the upside, but later as the wall moves upward and blocks the previous passage, it changes its path and passes through the space under the wall. Figure 6.7(a) shows the on-line execution time of the developed planner in this simulation case, where each data point denotes the time value needed to execute a certain type of operation in a single cycle, e.g., one point denoted by '□' means the time duration for the planner to find a collision-free path from the current configuration to the goal configuration at the current cycle; similarly, the '×' points refer to the time for superposition of the C-obstacle maps in a single cycle, and those '○' points correspond to the time for A* path searching in a single cycle. In Figure 6.7(a), when a point is located at the x axis, it means the time is less than 1 millisecond, since the accuracy threshold of the developed simulation program is 1 millisecond. The result shows that the on-line running time at each cycle is less than 150 milliseconds. Figure 6.7(b) gives the depth number until which the path searching ends in each planning cycle. Overall, the paths are found at the depth not more than 4. Furthermore, a strong dependence exists between this planning time and the depth number. Within each planning cycle, the operation of the C-obstacle map superposition accounts for most of the time while the A* path searching consumes far less time.

Chapter 6 Path Searching in Configuration Space

Figure 6.6 Dynamic simulation scenario- Screenshot of simulation case 2

(a) Planning time VS cycle number

(b) Ending depth VS cycle number

Figure 6.7 Result of the simulation case 2

6.3.3 Case 3 – Static and Dynamic Obstacles

In Section 4.2.4.2, a simulation scenario containing both static and dynamic obstacles is designed to test the developed W-space planner. Here, a similar scenario is adopted to test the real-time performance of the C-space planner. As shown in Figure 6.8, a static wall is fixed while several dynamic spheres are placed. These spheres wander in the workspace with random initial velocity and rebound from the boundary of the cuboid workspace when they hit it. During the simulation, the C-space corresponding to the whole environment at each planning cycle is composed of the part corresponding to the static wall and the part

corresponding to the dynamic spheres. The former is calculated once and preserved throughout the simulation, whereas the latter is updated at each planning cycle. The same *COD* settings as those in Section 6.3.1 are adopted.

The simulation result is given in Figure 6.8. Unfortunately, it is found in Figure 6.8(f) that under such a situation the developed planner is not sensitive of the approaching of dynamic obstacles to the robot, and collision occurs. This phenomenon can be explained by a 2D example as shown in Figure 6.9. In this figure, a configuration obstacle (C-obstacle) block is located between the robot (called C-robot) and its goal and this C-obstacle keeps moving towards the C-robot. The mission of the planner is thus to guide the C-robot to reach the goal safely. Normally, the motion planner tends to find out a feasible path adjacent to the C-obstacle and then the robot follows it, as shown in Figure 6.9(a). If the obstacle is static, such a strategy is safe and efficient. However in the current case, the C-obstacle keeps moving towards the C-robot. This leads to the situation in Figure 6.9(b) where the C-obstacle gets quite close to the C-robot. Finally as shown in Figure 6.9(c), collision occurs. This example demonstrates that the motion planner developed so far is not receptive to approaching dynamic obstacles towards the robot.

6.3.4 Case 4 - Simulation with Two Robots

As shown in Figure 6.9, a simulation case including two ESHED SCORBOT ER4pc robots is designed to simulate a more realistic robotic application, where one robot acts as a reference (on the right side) and moves freely in the environment, whereas the other robot (on the left side) needs to avoid possible collision with its partner. During the simulation, the base of the reference robot serves as a static obstacle, and the other links of this robot are regarded as dynamic obstacles. The same *COD* settings as those in simulation case 1 are adopted again. Applying the developed motion planner, some key screenshots are shown in Figure 6.10. A collision occurs between the two robots as demonstrated in Figure 6.10(f). This simulation case again reflects that the C-space planner is not receptive to dynamic obstacles approaching the robot.

Figure 6.8 Collision occurred between the robot arm and spheres

Figure 6.9 Demonstration of the non-sensitivity of the developed C-space planner

Figure 6.10 Simulation case including two robots

6.4. Conclusion

In this chapter, a practical on-line motion-planning approach based upon pre-computing the global connectivity in the configuration space with respect to all of the possible obstacle positions in the workspace is proposed. The planner is verified as effective for an industrial robot with five DOF in both a complex static environment and a simple dynamic environment.

Representing the C-space by a hierarchical data structure in format of non-uniform 2^m -tree and searching collision-free paths under the multi-resolution strategy has provided good efficiency to the planner. Furthermore, this C-space planning approach can provide resolution completeness.

It is also observed that the developed C-space planner is not receptive to dynamic obstacles approaching to the robot, which may cause unwanted collisions. To overcome this problem, the planner must be improved to respond rapidly when any obstacle gets too close, and this will be discussed in the next chapter.

Chapter 7 Compound Motion Planner

In the previous two chapters, a C-space planner for on-line motion planning was discussed. This planner consisted of an off-line stage and an on-line stage. In the off-line stage, a C-obstacle database (*COD*) storing the connectivity information in the C-space is pre-computed for a given manipulator. In the on-line stage, the C-space construction and path searching are interleaved under a multi-resolution manner. This C-space planner is complete in terms of resolution. However, one last remaining issue remains; it is not sensitive to moving obstacles approaching the robot. To avoid possible collision, the C-space planner must be enhanced further to avoid moving obstacles effectively so as to be finally a viable dynamic path planner.

In Section 4.2, a reactive approach resembling the potential field method is applied in the third version of the W-space planner and verified as effective and time efficient to avoid dynamic obstacles. Therefore, in this chapter we intend to incorporate this reactive method into the pure C-space planner.

7.1 Compound On-line Planning Design

A compound motion-planning approach is proposed here in which the global collision-free paths will be searched by using the C-space approach described in Chapter 6, and the reactivity to obstacles approaching the robot will be provided by employing the reactive method as described in Section 4.2.

On one hand, this compound planner combines the W-space planning with the C-space planning: the C-space construction and path searching is executed in the C-space, while the reactive module for local obstacle avoidance is implemented in the W-space. On the other hand, it integrates the global planning and local planning together, since the C-space planning belongs to the global planning category whereas the obstacle avoidance in the W-space is suitable for local planning.

The structure of the compound planner is shown in Figure 7.1 and further explained as follows. The joint change rate of a robot at $t = t_i$ is obtained through the superposition of the joint rates obtained from the W-space planning and the C-space planning respectively. According to Algorithm 6.2, the joint step change from the C-space searching at $t = t_i$ is denoted as

$$\Delta\theta_C = (\Delta\theta_{C1}, \dots, \Delta\theta_{Cn}) = Path(t_i, \Delta) \quad (7.1)$$

According to Equation 4.17, the joint rate coming from the module of obstacle avoidance at $t = t_i$ can be denoted as

$$\dot{\theta}_W = (\dot{\theta}_{w1}, \dots, \dot{\theta}_{wi}) = J_p^+ * (SF\left(\left|\frac{\vec{pq}}{pq}\right|\right) * \frac{\vec{pq}}{\left|\frac{\vec{pq}}{pq}\right|}) \quad (7.2)$$

Then, the total joint step change at t_i is

$$\Delta\theta = \Delta\theta_C + \dot{\theta}_W * T = Path(t_i, \Delta) + J_p^+ * (SF\left(\left|\frac{\vec{pq}}{pq}\right|\right) * \frac{\vec{pq}}{\left|\frac{\vec{pq}}{pq}\right|}) * T. \quad (7.3)$$

Again, the Jacobian matrixes appeared in the above equations can be referred to Appendix C.

7.2 Simulation

7.2.1 Simulation Case with Multi Spheres

As described in Chapter 6, in the simulation case including several dynamic spheres, collision may occur due to the insensitivity of the C-space planner to the dynamic obstacles. Here the same simulation scenario is again employed to test this compound on-line planner. The parameters, including the resolutions in the W-space and the C-space, are set to be the same as those in Chapter 6. The result of motion sequence is shown in Figure 7.2, which clearly shows that the robot can now move from the starting configuration to the destination progressively, without collision with both the static and dynamic obstacles.

Figure 7.1 The structure of the compound motion planner

(a)

(b)

Chapter 7 Compound Motion Planner Design

Figure 7.2 Screenshots in the simulation case including multi moving spheres

The joint trajectories of this simulation process are shown in Figure 7.3(a). The on-line execution time is summarized in Figure 7.3(b). The result shows that the on-line running time at each cycle is less than 150 milliseconds, mainly depending on the depth number at which the searching algorithm ends. Within each planning cycle, the operation of C-obstacle map superposition accounts for most of the time while the A* path searching consumes far less time, and the newly incorporated module for obstacle avoidance takes less than 1 millisecond.

(a) Joint trajectory

(b) Cycle planning time VS cycle number

(c) Ending depth VS cycle number

Figure 7.3 Planning results of the simulation case with multi moving spheres

7.2.2 Simulation Case with Two Robots

In this simulation case, the scenario described in Section 6.3.4 is retested using the compound motion planner. In Section 6.3.4, the pure C-space planner was tested but collision occurred in this scenario. According to the simulation settings, the main robot on the left side moves from the initial configuration to the goal as shown in Figure 7.4(a) to 7.4(h), and then returns from the goal to the initial configuration as in Figure 7.4(h) to 7.4(l); the reference robot meets the main robot at both trips and collision will occur if no evading movements are taken. The result generated after incorporating the module to avoid dynamic obstacle is given in Figure 7.4, which shows that the main robot can react fast enough and avoid the reference robot when the latter approaches it.

Chapter 7 Compound Motion Planner Design

Figure 7.4 Screenshot in the simulation case including two robots

The joint trajectories corresponding to the above motion sequence is depicted in Figure 7.5(a), where avoiding actions are observed in both trips. The time performance is summarized in Figure 7.5(b). The on-line planning time for all cycles still occur below 150 milliseconds.

(a) Joint trajectory of the main robot

(b) Cycle planning time VS cycle number

(c) Ending depth VS cycle number

Figure 7.5 Time performance of the simulation case with two robots

7.2.3 Simulation Case with Static Walls and Multi Dynamic Cylinders

In this simulation case, we try to compare the time performance of our planner with some existing results in [27]. In this reference, the path in a dynamic scene is searched based upon a roadmap that is pre-created for the static part of the scene, and the simulation result for an articulated robot with six DOF is provided. In our calculation, the same environment is adopted as that in [27]. Shown in Figure 7.6, the SCORBOT-ER4pc robot stands fixed amidst several static walls, and five moving cylinders hinder the robot in its attempt to go from a start configuration (robot fully bent to the right) to the goal configuration (robot fully bent to the left). Compared with the first simulation case in this chapter, here the higher environmental complexity is contained. We tested the developed compound planner in such an environment. Some screenshots of the result are presented in Figure 7.6, in which the robot is observed to successfully avoid both the dynamic cylinders and the static walls, and ultimately reach the destination at the left most position.

Figure 7.6 Screenshot in simulation case including static walls and dynamic cylinders

The corresponding joint trajectories are shown in Figure 7.7(a). The time performance is summarized in Figure 7.7(b). Under the adopted *COD* resolution, the on-line planning time at all cycles again distribute below 250 milliseconds. In [27], it took 0.87 s for their planner to compute the trajectory, which is more than 3 times longer than our result.

(a) Joint trajectory of the main robot

(b) Cycle planning time VS cycle number

(c) Ending depth VS cycle number

Figure 7.7 Planning Results of the simulation case including static walls and dynamic cylinders

7.2.4 Simulation with 7-DOF PA-10 Robot

In this simulation case, we verify the feasibility of the proposed compound planner for redundant manipulators. Once more, the Mitsubishi PA-10 manipulator with seven DOF is used as the prototype. Considering the high complexity in *COD* construction that may be caused by the high DOF of this manipulator, the decomposed strategy as described in Section 4.2 is again applied to simplify the motion planning problem. Similarly, we decompose the motion planning problem for this robot into two sub-problems, i.e., the inner links below the wrist serve to position the robot wrist center, and the joint axes outside the wrist determine the gripper orientation.

The implementation of these two sub-problems are similar, that is, in each sub-problem, a respective *COD* is setup, based on which the on-line path searching is executed, and the local obstacle-avoidance module is also called. In each planning cycle, planning is first executed for the position sub-problem and a new position of the wrist center is achieved; following this new position, the orientation sub-problem is then planned. The architecture of this planner is shown in Figure 7.8.

In this simulation case, the same testing environment with that in the previous case is adopted.

As shown in Figure 7.9, the PA-10 robot needs to go from the start configuration (robot fully bent to the right) to the goal configuration (robot fully bent to the left) without colliding with the static wall and the moving cylinders. In terms of geometric complexity, the PA-10 robot prototype is represented by a total of 1825 triangles, the static wall is decomposed into 4446 cells, and each moving cylinder is decomposed into 355 cells. The parameter settings to calculate the *COD* for the PA-10 robot are listed in Table 7.1.

Table 7.1 Parameter settings to calculate the *COD* for the PA-10 robot

		Position chain	Orientation chain
Joint collection		$I_P = \{1, 2, 3, 4\}$	$I_O = \{5, 6, 7\}$
W-space	Number of cells	64x64x64	64x64x64
	Resolution	31.25, 31.25, 20.7 mm	7.80, 7.80, 7.80 mm
C-space	Number of cells	$64 \times 32 \times 32 \times 16 = 10^6$	$32 \times 32 \times 32 = 3.2 \times 10^4$
	Resolution	5.6°, 6.0°, 11.2°, 17.9°	16.8°, 11.25°, 16.8°
<i>COD</i> Size		91.3 Mbytes	2.65 Mbytes

We tested the proposed compound planner and some screenshots from a successful running sequence are presented in Figure 7.9. The robot is observed to successfully avoid both the dynamic cylinders and the static walls, and eventually reach the destination at the left most position.

The joint trajectories corresponding to the above motion sequence is shown in Figure 7.9(a). It can be found from this figure that the joint values of the positioning links keep constant after $t \approx 35s$, which is due to our arranging the orientation sub-problem to start only when the position sub-mission is finished. The time performance of this simulation case is given in Figure 7.9(b). Under the adopted *COD* resolution, the on-line planning time for both sub-problems in most planning cycles is less than 400 milliseconds.

Figure 7.8 Flowchart of the compound planner for PA-10 Robot

Figure 7.9 Screenshots in simulation case 4 (with obstacle avoidance)

(a) Joint trajectory

(b) Planning time VS cycle number

Figure 7.10 Planning results of the simulation case 4

We also tried many different settings of cylinder velocities and cylinder initial positions in this testing scenario. Unfortunately, collisions were observed in some of them. A common

phenomenon when the planner failed is: the position chain finds a path for itself and moves safely towards its target; but at some positions on this path, the wrist center gets too close to a moving cylinder or the static wall, which causes the orientation chain to unavoidably hit the cylinder or the wall. This phenomenon also indicates that there is still an element of non-completeness introduced by the adopted decomposition strategy. Further research for this application is needed in future work.

For an on-line presentation, the demo videos of all the simulation cases in this thesis are also available via the following URL through the internet:

http://www.ntu.edu.sg/home5/pg03770495/research_robot.html.

The header file containing the declarations of those important functions of the simulation programs is given in Appendix E.

A summary of the developed motion planners in this thesis are given in Table 7.1.

Table 7.2 Comparison of the developed motion planners

	Main features	Time performance	Algorithmic complexity	Completeness	Suitable applications
W-space Planner Version 1	Multi-agent concept; Two-level hierarchical structure; Implementation by fuzzy logic	<1ms for a single planning cycle	Very low	weak	Simple static environments
W-space Planner Version 2	Vector-format fuzzy logic; Reactive obstacle avoidance	<1ms for a single planning cycle	Very low	weak	Simple dynamic environments with moving obstacles
W-space Planner Version 3	Decomposition strategy; Heuristic searching for the position sub-problem; Reactive obstacle avoidance	<10 ms for a single planning cycle in average	Low	Reduced resolution complete	Dynamic environments with medium complexity, high accuracy requirement
C-space Planner	Off-line <i>COD</i> setup; Non-uniform 2^m -tree representation of the C-space; multi-resolution path searching	Hundreds of milliseconds for a single planning cycle in average	Medium	Resolution complete	Complex static environments with medium or low accuracy requirement
Compound Planner	Combination of C-space searching and reactive obstacle avoidance in the W-space	Hundreds of milliseconds for a single planning cycle in average	Medium	Resolution complete	Dynamic environments with medium or low accuracy requirement

Chapter 8 Experimental Verification

To verify its feasibility in practical applications, the proposed compound motion planner is tested with a real robot platform in a dynamic environment in this chapter.

8.1 Introduction

To operate the robot safely in a dynamic environment, the developed on-line robot motion planner continuously senses the environmental changes and outputs updated robot trajectories accordingly. The robot controller then needs to drive the robot to track this time-varying trajectory. Figure 8.1 shows the constitution and data flow of a typical robot application, which demonstrates the interactions between the motion planning module, sensing sub-system and the motion controller.

Figure 8.1 Block diagram of a typical testing system for robot motion planner

The adopted testing hardware, the control algorithm and the testing results will be described in detail in this chapter.

8.2 Testing Hardware

Due to the limited available resources in the research lab, the SCORBOT ER4pc educational robot is chosen as the testing platform. The detailed specifications of this robot are given in Appendix A. The built-in control mechanism of this robot does not provide any interface for user interference. The companion control software only supports a specialized script language to program the robot motion, but does not support additional programming routines like C++ codes or .dll functions. Moreover, the robot does not have existing sensors to capture the time-varying environment.

To overcome these limitations and realize the communication between the planning module and the control module, the original control box of the robot is bypassed and a new interface between the host computer and the built-in servo motors attached to the robot joints is developed.

In this testing, we have not integrated the camera capturing module for the environmental sensing. Alternatively, a moving obstacle is created by pulling a rectangular thin plate in the vertical direction. The pulling belt is driven via a stepper motor. Using such a transmission facility, the position and speed of the plate can be controlled. Although no direct sensing facility is employed, the status of the plate can still be fed to the motion planner. The proposed testing system for motion planning is summarized in Figure 8.2.

Figure 8.3 shows the layout of the actual testing rig, and Table 8.1 lists the hardware components used for both the new interface and the moving obstacle sub-system.

Figure 8.2 Proposed motion control system

Figure 8.3 Layout of the actual testing rig

Table 8.1 Component list for experimental test

No.	Unit type	Model / Description
1.	Digital computer	Intel Pentium IV PC
2.	Data acquisition board	Humusoft MF 614 Multifunction I/O card
3.	Servo amplifier	AMC model 12A8 servo amplifier
4.	Power supply	Dual DC power supply (30V, 6A each)
5.	Pulse shaper for encoder	SN74LS244 TTL chip
6.	DC servo motors	Scorbot-ER 4PC built-in
7.	Encoders	Scorbot-ER 4PC built-in
8.	Micro-switches	Scorbot-ER 4PC built-in
9	Stepper motor	RS 440-420 unipolar stepper motor, 5.0v, 1.8deg step angle
10	Stepper driver	UCN5804 unipolar stepper-motor translator/driver

8.2.1 New Interface for Robot Control

Bypassing the original control box, the new interface acts as an alternative interconnection between the host PC and the SCORBOT ER4pc robot. Through this interface, the motion commands generated from the software controller are sent from the host PC to the motors; the motor status, like the encoder signals and the motor currents, are fed back from the motors to the host PC. The main hardware components of this interface are further introduced in the following section.

Humusoft MF 614 multifunction I/O card

An I/O card is needed to connect the Controller PC to real world signals. The Humusoft MF 614 Multifunction I/O card offers the following features:

- 4 independent 12 bit D/A converters with $\pm 10V$ output range.
- 4 quadrature encoder inputs with single-ended or differential interface with support up to 2 MHz input frequency.

- A 100 KHz throughput 12 bit A/D converter with sample and hold circuit.
- 8 bit TTL compatible digital input port
- 8 bit TTL compatible digital output port
- Internal clock and voltage reference
- Interrupt

Figure 8.4 Humusoft MF 614 multifunction I/O card

The MF614 card can be installed easily in any free PCI expansion slot of a PC. The card supports C/C++ programming language and all of its ports, like Digital-to-Analog (DA), Analog-to-Digital (AD), Digital Input (DIN)/output (DOU), Timer and etc., can be accessed via C/C++ commands.

The MF614 I/O card only contains four Digital-to-Analog (DA) channels, so only four motors can be controlled simultaneously. Since the movement of the fifth robot joint, referred to as wrist roll, does not influence the tip position of the end-effector and does not play a significant role in obstacle avoidance of the robot arm, it is not controlled through the MF614 I/O card for this experiment.

Amplifier

The built-in servo motors attached to the first four joints are driven by A-M-C 12A8 servo amplifiers. Figure 8.5 shows the assembled amplifiers. They accept 20V DC inputs from a DPS-1306AF dual DC power supply and output a 12V voltage in PWM format to the motors.

Figure 8.5 Four A-M-C 12A8 servo amplifiers driving the servo motors

8.2.2 Moving Obstacle Sub-system

A dynamic scenario similar to that in Section 6.3.1 is applied to test the planner performance in a real application. As shown in Figure 8.6, a rectangular thin plate with dimension of $320\text{mm} \times 240\text{mm}$ is initially located at the bottom position. This moving obstacle module includes a stepper motor and its driver, a supporting bracket, a pulley and a belt. Via the above settings, the rectangular plate can move between the upper and bottom ends in the vertical direction.

Figure 8.6 The moving obstacle

The control signal to trigger the stepper motor is sent from the digital-output (DOUT) channels of the MF614 I/O card to the UCN5804 chip. Figure 8.7 shows the circuit of this stepper driver.

(a) The circuit of the stepper driver

(b) Photo of actual stepper driver

Figure 8.7 The stepper driver

8.3 Control Algorithm and Software Implementation

8.3.1 Control Scheme

After the motion planner outputs a desired configuration, the motion controller needs to generate control actions in order to track the trajectory. In this experiment, a joint space control scheme is applied for this purpose. For each joint axis, a PD type controller is designed which leads to the following control output.

$$\tau = K_p * e(t) + K_D * \dot{e}(t) \quad (8.1)$$

where $e(t)$ and $\dot{e}(t)$ are expressed as counting numbers from the encoder readings, τ is the control voltage to the A-M-C amplifier. The values for K_p , K_d are chosen by trying different values for the two constants while repeatedly running the control system. Table 8.2 lists the parameters of the PD controllers driving the servo motors.

Table 8.2 Parameters of the PD controllers

Joint Axis No.	P	D
1	20.0	20.0
2	10.0	20.0
3	10.0	10.0
4	25.0	30.0

8.3.2 Software Development

The compound motion planner developed in Chapter 7 and the above PD type of control algorithm are implemented in a single software via C++ programming. Two C++ classes are defined, one for motion planning and the other for motion control. Appendix E gives the member variables and member functions of these two classes. Multithreading programming technique is utilized in the software development. Two threads are defined, running the planning module and control module respectively. The two threads are coordinated at each sampling interval.

8.4 Testing Results

The time-variant motion commands and the real robot motion are recorded in the experiment. The comparison between them shows a good consistency, which demonstrates the practical feasibility and efficiency of the proposed planners. In this testing case, the thin plate is set to move with a constant speed of 20mm/s. When arriving at the upper or bottom edge, the plate reverses its moving direction. It is also worth noting that at each planning cycle, the motion planner only makes use of the obstacle position information at the current moment; thus it does not know the future movement of the obstacles in advance.

8.4.1 Comparison of the Calculated and the Real Robot Motion

As illustrated in Figure 8.8, during the running process of motion planning, the motion of the virtual robot in the visualization window is synchronized with the movement of the real robot. Figure 8.9 lists some snapshots of the testing process. The right column presents the intermediate shots from the simulation window, and the left column shows the real robot configurations occurring at the same moments as the snapshots in the right column. The initial and goal configurations are set as

- initial configuration in degree: (-60, 0, 50, -110,0)
- goal configuration in degree: (60, 10,30, -100,0)

The rotation range of the first joint is (-130, 180) in degree, which means the robot must pass the intermediate configuration with $\theta_1 = 0^\circ$ in order to arrive the destination. According to the scenario setting in Figure 8.8, the thin plate just locate between initial and the goal robot configurations.

Figure 8.8 Synchronized real and visualized robot motion

(a)-1

(b)-1

(a)-2

(b)-2

Chapter 8 Experimental Verification

(a)-3

(b)-3

(a)-4

(b)-4

(a)-5

(b)-5

Chapter 8 Experimental Verification

(a)-6

(b)-6

(a)-7

(b)-7

(a)-8

(b)-8

(a)-9

(b)-9

(a)-10

(b)-10

Figure 8.9 Comparison of the calculated motion sequences and the real robot motion

(Column (a) refers to the real robot, Column (b) denotes the simulated robot)

As can be observed from the above motion sequence, the thin plate is initially located at the bottom and the collision-free path found by the planner gets around the plate from its upper side. Latter as the plate moves up, the previous passage gets blocked by the moving plate; then, the planner generates a different path following which the robot end-effector passes from beneath the plate (Figure 8.9.a(4)) and continues to move till the goal is reached.

8.4.2 Motor Trajectories

The joint trajectories and errors of the four joints are shown in Figure 8.10-8.13.

(a) Joint trajectory

(b) Joint error

Figure 8.10 Joint trajectory and joint error of the 1st axis

(a) Joint trajectory

(b) Joint error

Figure 8.11 Joint trajectory and joint error of the 2nd axis

(a) Joint trajectory

(b) Joint error

Figure 8.12 Joint trajectory and joint error of the 3rd axis

(a) Joint trajectory

(b) Joint error

Figure 8.13 Joint trajectory and joint error of the 4th axis

The demo videos of this experiment to prove the process are also available via the following URL through the internet:

http://www.ntu.edu.sg/home5/pg03770495/research_robot.html.

8.4.3 Comments of the Testing Result

The result observed in this testing verifies that the developed motion planner continuously generates collision-free paths according to the moving obstacle, until the robot successfully arrives at the preset destination. Also, the recorded motion sequence and the joint trajectories demonstrate that the real robot can closely follow the time-varying path generated by the motion planner. In summary, the testing fully proves the practical feasibility of the proposed motion planner for a real robot in the created dynamic environment.

The testing executed in this chapter can be further enhanced on some aspects. First, observed from the recorded videos, the real robot motion is not too smooth, which are mainly due to the following reasons. The SCORBOT robot uses a timing belt transmission, which limits the response and accuracy of the robot. More importantly, the fairly simple PD control algorithm employed for each joint axis during this testing, whose parameters are determined by trial-and-error, may not be optimal. Therefore, an accurate analytic model reflects the robot dynamics and a more advanced control algorithm can lead to a better controller design and help to achieve better robot motion with faster response.

It is also noted that the created dynamic environment forms the basis of a good test case example to prove the ability of the path planner. For future work, more complicated dynamic environments can be designed for testing as the proposed motion planner has the potential to deal with them when provided with a powerful sensing system.

Chapter 9 Conclusion

9.1 Work Accomplished in this Research

This PhD research focuses on the development of practical on-line motion planners for typical industrial robots in unstructured dynamic working environments. Aiming at handling both quasi-dynamic and dynamic environmental changes with less computational cost or time, several effective and efficient motion planners have been developed and verified through analytic and simulation studies in this thesis. The work accomplished in this research is summarized as follows:

9.1.1 Planning in the Workspace

Robot motion planning in the workspace (W-space) can achieve low computational complexity, which arises from the simplicity in describing the positional relationship between the robot and environment. Following this planning framework, three versions of the W-space planner have been developed in the sequence as described in this thesis, each with improved performances.

In the first version, each robot link, including the end-effector, is considered as an agent, which executes one behavior selected from a predefined behavior collection at every planning cycle. There is no role or priority distinction among these agents. A two-level hierarchical structure is then designed to carry out the motion planning. The higher level is designed to dynamically assign each link an appropriate behavior and the lower level is designed to determine the joint speed according to the behaviors assigned by the higher level. In this hierarchical structure, both levels are implemented based on fuzzy reasoning. Due to the hierarchical planner architecture and the low computational cost from using fuzzy reasoning, a single planning cycle can thus be implemented almost instantly (in less than 1 millisecond). A back-tracking mechanism is further incorporated into the planner, trying to escape local minima. Simulation studies in two scenarios with a single static obstacle placement have

verified the feasibility of the proposed planner. The main limitation of this planner version is that the introduced back-tracking mechanism does not always ensure an effective escape from local minima and is not suitable for dynamic working environments.

The second version of the W-space planner is developed to overcome the limitations of the first version motion planner. A different planner architecture is introduced, following which the end-effector guides the whole robot arm towards the goal, and a novel vector-format fuzzy reasoning approach is proposed in this thesis in order to implement the task of guiding the end-effector. This new fuzzy reasoning approach can manipulate both scalar and vector type of variables in a compact way, and hence reduce the size of the fuzzy rule-base. A set of new membership functions to manipulate fuzzy vectors is defined, and a series of new fuzzification, fuzzy inference and defuzzification approaches in vector-format is also developed. On the other hand, following this new planner architecture, the left robot links other than the end-effector are assigned to avoid local obstacles. For this purpose, a reactive method resembling the potential-field approach is developed. When an obstacle gets close to the robot, the robot link which is closest to the obstacle will reactively move away from the latter with a moving speed in reverse proportion to the minimum distance. This required positional change of the link is further transformed into joint movements via general-inverse Jacobian.

Simulation studies verify that, in a scenario containing a single moving obstacle, this new planner version succeeds to drive the robot prototype to reach a preset destination, meanwhile keeping the whole robot arm safely away from the obstacle. In another simulation scenario which is a typical case of local minimum, the new planner succeeds in finding a path for the robot to go around a large wall. Compared with the previous planner design, the second version can deal with dynamic obstacle by discarding the backtracking mechanism; furthermore, its capability to escape local minima is also improved, mainly attributed to the new planner architecture and the introduction of a new behaviour *move_around* which is specially designed for the end-effector to move around obstacles in the vector-format fuzzy reasoning. The limitation of this planner version is that it may fail to guide the robot to

traverse in more complex scenarios, such as a static non-convex environment. It is also noted that the proposed vector-type fuzzy reasoning to guide the robot end-effector is still a trial-and-error method.

In the third version of the W-space planner design, based upon the wrist-decouple structure of typical industrial robots, the overall motion planning problem is decomposed into the position part and the orientation part, each with a lower DOF than the overall one. In the first sub-problem, the robot workspace is represented by a hierarchical tree structure, in which the continuous trajectory is searched for the robot wrist-center by applying a global searching algorithm; in the second sub-problem, following the trajectory found in the previous sub-problem, the pose of the gripper is locally adjusted to approach the final configuration.

This third version of W-space planner can find a path more confidently than the previous trial-and-error approaches, and also reduces the occurrence of local minima considerably. In simulation studies, the static non-convex scenario in which the second planner version fails is retested and this third version is demonstrated to be able to lead a robot arm to traverse through it successfully. Moreover, this planner succeeds in navigating the robot through a scenario including a static wall and multiple moving obstacles. The main limitation for this design is that it is not complete so that when the robot follows a collision-free path for the wrist center, the robot links are not guaranteed to be collision-free.

9.1.2 Planning in the Configuration Space

In the second stage of this PhD research, the configuration space (C-space) planner is developed in order to seek complete solutions for on-line motion planning. A practical on-line robot motion planning approach that is based upon pre-computing the global C-space connectivity with respect to all possible obstacle positions is proposed.

The proposed motion planner consists of an off-line stage and an on-line stage. In the off-line stage, the obstacles in the C-space (C-obstacle) with respect to the obstacle positions in the

workspace are computed, which are then stored using a hierarchical data structure with non-uniform 2^m -trees. In the on-line stage, the real obstacle cells in the workspace are identified and the corresponding 2^m -trees from the pre-computed database are superposed to construct the real-time C-space. The collision-free path is then searched in this C-space by using the A* algorithm under a multi-resolution strategy which has excellent computational efficiency.

The proposed approach belongs to the category of global planning, and the whole environmental information is taken into consideration during the planning process. Given a preset resolution, this approach ensures that a collision-free path can definitely be found if it does exist. Compared with the W-space planners developed earlier, this proposed C-space planning method can thoroughly avoid local minima. By using the hierarchical C-space representation and multi-resolution strategy, the searching goes into higher resolution only if no path is available at the current resolution level, thus the on-line computational time and memory can be significantly reduced. Simulations on the prototype – the ESHED SCORBOT ER4pc Robot in a static scenario with challenging narrow passages prove the planner's ability of global localization. In another simulation scenario containing a moving wall, the path can be found at a coarse resolution with quite a low time requirement. In the 3rd simulation case, multi moving obstacles are placed in the scenario and for the 4th simulation case, two co-operating robots are included; however, collision occurs in both cases due to the developed C-space planner not being sensitive to moving obstacles approaching the robot. To overcome this limitation will be the objective of the next design.

9.1.3 Compound Planner Design

The C-space planner developed previously is resolution complete but lacks sensitivity to dynamic obstacles; whereas the reactive method applied in the W-space planner is verified as highly reactive to changes of local obstacles and computationally efficient. This reactive module is thus incorporated into the C-space planner and a compound motion planner is developed. By such a compound planner, the global collision-free path is searched in the

C-space by the A* searching algorithm, and the reactivity to local obstacles approaching the robot is provided by the reactive method executed in the W-space. The two simulation scenarios in which the pure C-space planner fails are retested to verify the compound planner, and the results clearly demonstrate that by applying the compound planner, not only can the collision-free path be continuously found, but the robot also reacts fast enough to avoid other approaching obstacles.. Lastly, the compound planner is tested in a complex dynamic scenario similar to the work in [27] employing both robot prototypes. A better time performance is observed from the simulation result.

9.1.4 Experimental Verification

To move beyond simulated results, the path planner feasibility is verified in practical applications. The compound motion planner is tested with a real robot platform in a dynamic environment. In the experiment, the SCORBOT ER4pc educational robot is chosen as the testing platform; the original control box of the robot is bypassed and a new interface is setup between the host computer and the built-in servo motors attached to the robot joints; a moving obstacle is created by pulling a rectangular thin plate in the vertical direction. In order to track the trajectory generated by the motion planner, a joint space control scheme is applied and PD type of controllers are designed for each joint axis.

The testing result verifies that the developed motion planner continuously generates collision-free paths according to the moving obstacle, until the robot successfully arrived at the preset destination. It is also observed that the real robot closely follows the time-varying path generated by the motion planner.

9.2 Contributions

The contributions of this thesis which expands further on the knowledge of dynamic path planners are summarized as follows:

- ♦ **A hierarchical planner architecture is designed to achieve on-line motion planning of**

manipulators. In this planner, the robot is considered as a multi-agent system, and both the higher-level and lower-level planners in the hierarchical design are implemented by fuzzy reasoning. The time requirement of this planner version at each planning step is low, which arises from the hierarchical structure design and distributing the planning task into the sub-motion of each link.

- ◆ **A vector-format fuzzy reasoning is proposed.** This new fuzzy reasoning approach can manipulate both scalar and vector types of variables in a compact way, and hence reduce the size of the rule-base. A set of new membership functions for manipulation of the fuzzy vector variables is defined and a series of new vector-format fuzzification, fuzzy inference and defuzzification approaches are also developed. This vector-format fuzzy reasoning approach is verified as being effective in guiding the robot end-effector in W-space.

- ◆ **An advanced W-space planner is developed.** This planner decomposes the overall planning problem into the position part and the orientation part with lower DOF. The collision-free path is searched in the W-space using a global searching algorithm, whereas the robot links can avoid local obstacles effectively using a reactive method resembling the potential-field approach. This planner has been proven to give excellent time performance, and is especially suitable for real-time motion planning of typical industrial robotic applications with large clearance between robots and obstacles.

- ◆ **Pre-computing the global configuration space connectivity.** With respect to all of the possible obstacle positions in the workspace, a real-time C-space construction is proposed to realize this. The global connectivity information of a given robot is required to be created only once and can subsequently be used in different environments. This construction approach is general and thus is applicable to manipulators of any kinematics structure and geometric shape. By this approach, the most time-consuming operation is implemented in the offline stage, and the online computing only needs to account for the real-time obstacles presented in the environment; therefore, high computational efficiency for on-line planning is achieved.

- ♦ **Heuristics including representing the C-space by hierarchical non-uniform 2^m -tree and a multi-resolution searching strategy.** This is applied to path planning in an m-dimensional C-space. By applying the non-uniform 2^m -tree data structure, the required storage space can be reduced considerably; under the multi-resolution strategy, collision-free path can be found at a coarse resolution with a low time cost.
- ♦ **A compound motion planner, which is computationally efficient and resolution-complete, is developed.** By this compound planner, the global collision-free path is found in the C-space by a fast searching algorithm, and the rapid reactivity to moving obstacles approaching the robot is provided by reactive method resembling the potential-field approach.

9.3 Future Work

To further improve the developed motion planners, the main future work is highlighted and recognized to make them even more robust and practical.

Improvement on Vector-format Fuzzy Reasoning Approach

The vector-format fuzzy logic motion planning approach can be further improved. In this research, only the distance and pointing are taken as the fuzzy inputs. In future work, other variables describing the motion of the manipulator, such as the relative speed between obstacles and the robot, can be added as the inputs to the fuzzy planner to improve the performance.

Introduce Adaptability to Fuzzy Planner

In the current work, the rule bases and the fuzzy membership functions in the fuzzy planners are pre-defined and not changeable, which means the path can be found and determined but the efficiency has not been taken into consideration. In order to overcome this problem, the rule bases and the membership functions needs to be modified by an adaptive approach based

on the actual situation of the manipulation environment, such as reinforcement learning based on neural network algorithms, to enhance the motion efficiency. For example, when the manipulator is far away from the goal and obstacles, the membership functions can be adjusted to speed up the motion; when the robot is close to obstacles, the membership functions should be adjusted to achieve fine motions.

Consider the joint accelerations in the fuzzy rule base

For dynamic environment planning, it is very important to reflect the dynamics/force information in the robotic system. For this purpose, the acceleration terms need to be incorporated in the fuzzy rule base.

COD Creation and Access for Manipulators with Higher DOF

The proposed C-space construction approach makes use of discretized representation of the C-space. In the simulation studies, it is observed that, the required space to store the *COD* increases drastically as the number of DOF and resolution increase. Moreover, in Chapter 5 the *COD* is created in a bottom-up fashion, which is easy to implement but quite time-consuming. So far, only manipulators with $\text{DOF} \leq 6$ can be practically stored. A novel C-space representation scheme with less space requirement and with less time cost to build is needed for applications involving robots with higher DOF. Possible solutions are given as follows.

- In [40] and [111], a top-down fashion based method has been applied to construct 2^m -trees and verified more efficient than the approach in bottom-up fashion. This method is also applicable to construct the C-obstacle maps and thus reduce the time for off-line *COD* creation. To realize the non-uniform 2^m -tree construction in a top-down fashion, an efficient algorithm is needed to detect collisions between a three-dimensional entity and a three-dimensional swept volume. To develop this algorithm will be one of the future tasks.
- In the current simulation scenarios, the obstacles in W-space are decomposed uniformly, which leads to a large number of cells. For instance, the “wall” in Figure 6.4 is

decomposed into a total of 1840 cells. If the obstacles in W-space are also decomposed hierarchically, for example, by octree representation, the total number of obstacle cells can be considerably reduced. If the *COD* is correspondingly organized in such a hierarchical way, the time needed for C-obstacle map superposition can be further shortened considerably. To represent and store the C-space connectivity using hierarchical roadmaps is also a possible research direction.

- Some commercial database can be employed to store the *COD* and provide faster record access, and help to achieve faster on-line motion planning.
- The development in hardware, especially in high-speed and large-scale storage equipment, also helps to improve the creation, read/write of *COD* maps. For example, recently high-speed flash hard drive reaches 155GB [134].

Fast Path Searching

Currently, path-searching is performed in a single direction. To speed it up, instead of searching from the start to the goal, two searches can be executed in parallel, with one from the start to the goal and the other from the goal to the start. When these two paths meet, a collision-free path can be found. Instead of choosing the best forward-search cell, i.e., $[g(\text{start}, x), h(x, \text{goal})]$ or the best backward-search cell, i.e., $[g(y, \text{goal}), h(\text{start}, y)]$, this algorithm will choose the best pair of cells, i.e., $[g(\text{start}, x), h(x, y), g(y, \text{goal})]$.

Testing the Motion Planners with Real Robot in More Complicated Environments

The experimental verification in Chapter 8 which was implemented could have been set with a more complex dynamic environment situation if not due to the constraints of the available hardware facilities. In the future work, more complex dynamic environments can be designed to test the planners especially if tested for a multi-robot application which commonly exists in many current industrial applications. Sensing system like optical sensors can also be integrated to provide a reliable method for environment detection.

Appendix

Appendix A - Specification of the SCORBOT ER4pc Robot

The SCORBOT-ER 4pc robot is a versatile system for educational purpose [135]. As shown in Figure A.1, this robot arm is mounted on a pedestal base. The high speed and the repeatability of the design make the robot suitable for both stand-alone operations and integrated applications in automated workcells such as robotic welding, machine vision, CNC machine tending and other flexible manufacturing system (FMS) operations. The robot is designed to enable observation of its working mechanical parts while ensuring a safe environment for students.

Figure A.1 Appearance of the SCORBOT Robot arm and its control box

Mechanical Specifications

The specification of the SCORBOT robot is listed in Table A.1.

Table A.1 Mechanical specifications of ESHED SCORBOT ER4pc Robot

Mechanical structure	<ul style="list-style-type: none"> Vertically articulated
Degrees of freedom	<ul style="list-style-type: none"> 5 rotational axes + gripper
Payload capacity	<ul style="list-style-type: none"> 2.1 kg (4.6 lb)
Axis Range	<ul style="list-style-type: none"> Axis 1: Base rotation: 310° Axis 2: Shoulder rotation: $+130^{\circ} / -35^{\circ}$ Axis 3: Elbow rotation: $\pm 130^{\circ}$ Axis 4: Wrist pitch: $\pm 130^{\circ}$ Axis 5: Wrist roll: Unlimited (mechanically); $\pm 570^{\circ}$ (electrically)
Maximum operating radius	<ul style="list-style-type: none"> 610mm (24") end of gripper
Maximum path velocity	<ul style="list-style-type: none"> 700 mm/sec (27.6"/sec)
Standard gripper	<ul style="list-style-type: none"> Servo motor, parallel fingers
Gripper opening	<ul style="list-style-type: none"> 75 mm (3") without pads; 65 mm (2.6") with pads
Repeatability	<ul style="list-style-type: none"> ± 0.18 mm (0.007")
Feedback	<ul style="list-style-type: none"> High resolution incremental optical encoder on each axis and gripper
Homing	<ul style="list-style-type: none"> Microswitch home on each axis
Actuators	<ul style="list-style-type: none"> 12VDC servo motor on all axes and gripper
Transmission	<ul style="list-style-type: none"> Gears, timing belts, lead screw
Weight	<ul style="list-style-type: none"> 10.8 kg (23.8 lb)
Ambient operating temp.	<ul style="list-style-type: none"> 2°-40°C (36°-104°F)
Additional features	<ul style="list-style-type: none"> Roller bearing support on all axes Anti-backlash gearing system on base axis Robot connects to controller through single 50-pin cable Built-in pneumatic cabling enables use of pneumatic end effectors

Special driving architecture requiring kinematic compensation

The SCORBOT ER4pc robot employs a special transmission mode, that is, when the shoulder joint is driven to move, the value elbow joint will also change rather keeping constant, while the motor driving the forearm keep static and with the upper arm's orientation keep unchanged. This phenomenon comes from the transmission design of the SCORBOT robot. One consequence of this design is that the angle of the elbow joint does not uniquely correspond to the driving motor's position, which is normally in terms of counting number from the encoder.

Therefore, during the control implementation, compensation is required to get rid of the effect of this co-changing of the joint. For this purpose, such a simple compensation scheme is adopted. It is observed that as the upper arm moves a certain angle, say, the shoulder joint θ_2 changes $\Delta\theta_2$, the change of the elbow joint θ_3 will just equals $-\Delta\theta_2$ too, i.e., $\theta_3 = \theta_3 - \Delta\theta_2$. And the same observations exist between the elbow joint and the wrist pitch joint.

Forward Kinematics

The link coordinates of SCORBOT ER4pc robot is shown in Figure A.2 and the D-H parameters of this robot are listed in Table A.2.

Figure A.2 Link coordinates of ESHED SCORBOT ER4pc Robot

Table A.2 D-H parameters of SCORBOT Robot

Axis	θ	$d(\text{mm})$	$a(\text{mm})$	α
1	θ_1	d_1	a_1	-90°
2	θ_2	0	a_2	0
3	θ_3	0	a_3	0
4	θ_4	0	0	-90°
5	θ_5	d_5	0	0

Inverse Kinematics

The tool-configuration vector for this five-axis articulated arm is:

$$W(q) = \begin{pmatrix} C_1(a_1 + a_2 C_2 + a_3 C_3 - d_5 S_{234}) \\ S_1(a_1 + a_2 C_2 + a_3 C_3 + d_5 S_{234}) \\ d_1 - a_2 S_2 - a_3 S_{23} - d_5 C_{234} \\ -\exp(q_5 / \pi) C_1 S_{234} \\ -\exp(q_5 / \pi) S_1 S_{234} \\ -\exp(q_5 / \pi) C_{234} \end{pmatrix} \quad (\text{A.1})$$

Base joint

$$q_1 = a \tan 2(w_2, w_1) \quad (\text{A.2})$$

Elbow joint

$$q_{234} = a \tan(-(C_1 w_4 + S_1 w_5), -w_6) \quad (\text{A.3})$$

define $b_1 = C_1 w_1 + S_1 w_2 + d_5 S_{234}$

$$b_2 = d_1 - a_4 S_{234} - d_5 S_{234} - w_3$$

then after rearrangement

$$b_1 = a_2 C_2 + a_3 C_{23}$$

$$b_2 = a_2 S_2 + a_3 S_{23}$$

define $\|b\| = a_2^2 + 2a_2 a_3 C_3 + a_3^2$

$$q_3 = \pm \arccos \frac{\|b\|^2 - a_2^2 - a_3^2}{2a_2 a_3} \quad (\text{A.4})$$

Shoulder joint

$$q_2 = a \tan(a_2 + a_3 C_3)b_2 - a_3 S_3 b_1, (a_2 + a_3 C_3)b_1 + a_3 S_3 b_2) \quad (\text{A.5})$$

$$q_4 = q_{234} - q_2 - q_3 \quad (\text{A.6})$$

Tool roll joint

$$q_5 = \pi \ln(w_4^2 + w_5^2 + w_6^2)^{1/2} \quad (\text{A.7})$$

Appendix B - Specification of PA-10 Robot

PA-10 is a 7-DOF redundant robot, widely used in industrial applications. Compared with SCORBOT ER4pc, PA-10 has 2 additional degrees-of-freedom, i.e. S3 and E2 as shown in Fig. 4.11, which both denote DOF rotating around the link.

Figure B.1 Mechanical Configuration of the PA-10 Manipulator

Mechanic Specification

Table B.1 Mechanical Specifications of PA-10 Robot

Main Body			
Model		7 degrees of freedom	
Mass of main body		40kg	
Load capacity		10kg	
Joint operating range		S1(rotation)	+/-177 deg
		S2(swing)	-64 to +124 deg
		E1(swing)	+/-174 deg
		S3(rotation)	-107 to +158 deg

	E2 (rotation)	+255 deg
	W1(swing)	+165 deg
	W2(rotation)	+255 deg
Max.Operating speed	S1,S2 axis	57 deg/s
	S3,E1axis	114 deg/s
	E2,W1,W2 axis	360 deg/s
Cleanliness		Class 1(0.2micrometer)
Configuration		Dust-proof/drop-proof construction(IP54)
Ambient environment		Temperature 0 to 50 ° C, Humidity under 90%RH(no condensation)
Drive method, Brake		AC servo motor, Electromagnetic brake
Position Sensor		Absolute angle sensor(resolver)
Positional repeatability		+0.1mm
Max length between joint		930mm
Air tubes for the user		2(Outside diameter 4mm,inside diameter.5mm)
Signal lines for the user		6(0.3sq)、3(0.75sq)
Controller		
Trajectory control		Point-to-point(PTP) (line interpolation, circular arc interpolation, circle interpolation, each axis interpolation)
External input/output signals		Input : 24CH、 Output : 24CH
External dimensions		240(W)×400(D)×300(H)mm
Mass		18kg(except motion control CPU board,PC,pendant)
Power supply		1P AC100-240V±10% 50/60Hz under 1.5kVA
Ambient environment		Temperature0 to 40 ° C Humidity under 80%RH (no condensation)
Motion control CPU board		Correspondence to PCI bus (insert into the PCI bus of the PC)
Digital input/output board		Correspondence to PCI bus(insert into the PCI bus of the PC)

Kinematics

The link coordinates of PA-10 robot are shown in Fig. 4.12 and the D-H parameters of this robot are listed in Table 4.5.

Figure B.2 Link Coordinates of PA-10

Table B.2 Kinematics Parameters (D-H Representation) of PA-10 Robot

Axis	θ	θ_0	d (mm)	a (mm)	A
1	θ_1	0	$d_1=317$	0	-90°
2	θ_2	0	0	0	90°
3	θ_3	0	$d_3=450$	0	-90°
4	θ_4	0	0	0	90°
5	θ_5	0	$d_5=480$	0	-90°
6	θ_6	0	0	0	90°
7	θ_7	0	$d_7=70$	0	0

Appendix C - Calculating General-inverse Jacobian

B.1 Pseudo-inverse Jacobian based upon the End-Effector of the SCORBOT ER4pc Robot

From [136], the velocity state of end-effector can be expressed in matrix form in terms of the joint rates as follows:

$$\dot{X} = \begin{bmatrix} v_n \\ \omega_n \end{bmatrix} = J\dot{q} \quad (\text{B.1})$$

where $\dot{X}$ denotes the end-effector velocity, which consists of the linear component v_n and the angular component ω_n ; $\dot{q}$ is the joint speed, $J = [J_1, J_2, \dots, J_n]$ is the robot Jacobian matrix. In this thesis, the Jacobian matrix with respect to a point p in the i -th link ($0 < i \leq n$) rather than the end-effector is used for local obstacle avoidance in Chapter 4 and Chapter 7.

$$\dot{X}_r = \begin{bmatrix} v_r \\ \omega_r \end{bmatrix} = J_r \dot{q}_{1 \dots n} \quad (\text{B.2})$$

Figure C.1 Illustration of link frames and reference point

Refer to Figure B.1, assuming that point p has local coordinate (p_x, p_y, p_z) with respect to the n -th link frame, and the manipulator only involves revolute joints, the Jacobian matrix J_r

with respect to point p can be computed as in [136] as

$${}^0J_r = \begin{bmatrix} [{}^0z_0 \otimes {}^0P_{0,n}^*] \cdots [{}^0z_{i-1} \otimes {}^0P_{i-1,n}^*] \cdots [{}^0z_{n-1} \otimes {}^0P_{n-1,n}^*] \\ {}^0z_0 \quad \cdots \quad {}^0z_{i-1} \quad \cdots \quad {}^0z_{n-1} \end{bmatrix} \quad (B.3)$$

where ${}^0z_{i-1}$ is a unit vector along the (i-1)-th joint axis, and ${}^0P_{i-1,n}^*$ is a vector defined from the origin of the (i-1)-th link frame, O_{i-1} , to the reference point p located in the k-th link frame F_n . Please note Equation B.3 is expressed in the fixed frame, and the items can be calculated as

$${}^0z_{i-1} = {}^0R_{i-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad (B.4)$$

$${}^0P_{i-1,n}^* = {}^0p - {}^0O_{i-1} = T_0^n \cdot [p_x, p_y, p_z, 1]' - T_0^{i-1} \cdot [0, 0, 0, 1]' \quad (B.5)$$

with ${}^0R_{i-1}$ being the rotation matrix of the (i-1)-th link frame, T_0^n and T_0^{i-1} being the transformation matrix of the n-th and the (i-1)-th link frame respectively.

Alternatively the Jacobian matrix J_r can be obtained by direct partial differentiation as

$$J_r = [J_{r1}, \dots, J_{ri}, \dots, J_{rn}] = \begin{bmatrix} \frac{\partial({}^0p)}{\partial\theta_1} & \cdots & \frac{\partial({}^0p)}{\partial\theta_i} & \cdots & \frac{\partial({}^0p)}{\partial\theta_n} \\ z_0 & \cdots & z_{i-1} & \cdots & z_{n-1} \end{bmatrix} \quad (B.6)$$

Please note Equation B.6 is also expressed in the fixed frame and similarly,

$${}^0p = T_0^k \cdot [p_x, p_y, p_z, 1]'.$$

It is worth noticing that Equation B.2 is a general expression from which the Jacobian with respect to any point in the k-th robot link ($0 < k < N$, n is the total number of joints) can be calculated; especially, when the tip of the end-effector is chosen as the reference point p will have local coordinate (0, 0, 0) and $k = N$.

Since the above Jacobian J_r may be not a square matrix, the *Moore-Penrose inverse* (variously known as the generalized inverse, pseudoinverse) [137] is thus computed as

$$J_r^+ = \begin{cases} J_r^T [J_r J_r^T]^{-1} & m \leq n \\ J_r^{-1} & m = n \\ [J_r^T J_r]^{-1} J_r^T & m > n \end{cases} \quad (B.7)$$

and $\dot{q} = J_v^+ \dot{X}$ gives the least-square solution of $\dot{q}$ and minimizes $\| \dot{X} - J \dot{q} \|^2$. The pseudo-inverse method is widely discussed in the literature but it often performs poorly because of instability near singularities.

Rather than just finding the minimum vector $\Delta\theta$ that gives a best solution to the equation $\Delta X = J\Delta\theta$, we find the value of $\Delta\theta$ that minimizes the quantity $\|J\Delta\theta - \Delta X\|^2 + \lambda^2 \|\Delta\theta\|^2$ [138], where $\lambda \in \mathbb{R}$ is a non-zero damping constant. Based on the above criteria, the damped least square solution is equal to

$$J_r^+ = \begin{cases} J_r^T [J_r J_r^T + \lambda^2 I]^{-1} & m \leq n \\ [J_r^T J_r + \lambda^2 I]^{-1} J_r^T & m > n \end{cases} \quad (B.8)$$

Applying the Singular Value Decomposition (SVD) method, the Damped Least Square solution in Equation B.8 for $J_r * \dot{\theta} = \dot{X}$ can be calculated in a very convenient way.

According to the SVD theory, the Jacobian matrix J_r can be decomposed as

$$J_r = UDV^T \quad (B.9)$$

Where U and V are orthogonal matrices and D is diagonal. If J is $m \times n$, then U is $m \times m$, D is $n \times n$, and V is $n \times n$. Then the damped least square solution in Equation B.8 is

$$J_r^+ = VEU^T \quad (B.10)$$

where E is the $n \times n$ diagonal matrix with diagonal entries equal to

$$e_{i,i} = \frac{d_{i,i}}{d_{i,i}^2 + \lambda^2} \quad (B.11)$$

where $d_{i,i}$ is the diagonal entries of D.

The advantage of this method is that a solution can still be achievable even near singularity.

Based on the above inverse Jacobian matrixes, an iterative numerical method [139] is adopted to compute the joint change.

- 1. Initialize joint variable q by the current joint configuration q_c , and the desired end-effector position is calculated as $X_d = X_c + dX$ where the positional change of the end-effector, dX , is the output of the global searching module.
- 2. Calculate the Jacobian matrix $J_r(q)$ and pseudo-inverse Jacobian J_r^+ .
- 3. Calculate the joint variation $dq = J_r^+ dX$.
- 4. Update the current joint configuration $q = q + dq$.
- 5. Calculate the new location of the end-effector frame corresponding to q , X_c , using the forward kinematics relationship.
- 6. Check whether X_c is the required end-effector position. If $dX_p = X_d - X_c$ is small enough, then stop the calculation, else go to step 2.

B.2 Pseudo-inverse Jacobian based upon the Links of the SCORBOT ER4pc Robot

For the ESHED SCORBOT ER4pc robot, the sub-Jacobian matrix J_p is given below. The reference point p with local coordinate (x, y, z) may be located at the k -th link, and $k=1, \dots, 5$.

$k=1,$

$$J_p = \frac{\partial p}{\partial \theta_1} = \begin{bmatrix} -xS_1 - zC_1 - a_1S_1 \\ xC_1 - zS_1 + a_1C_1 \\ 0 \end{bmatrix};$$

$k=2,$

$$J_p = \frac{\partial p}{\partial (\theta_1, \theta_2)} = \begin{bmatrix} -xS_1C_2 + yS_1S_2 - C_1z - a_2S_1C_2 - a_1S_1 & -xC_1S_2 - yC_1C_2 - a_2C_1S_2 \\ xC_1C_2 - yC_1S_2 - S_1z + a_2C_1C_2 + a_1C_1 & -xS_1S_2 - yS_1C_2 - a_2S_1S_2 \\ 0 & -xC_2 + yS_2 - a_2C_2 \end{bmatrix};$$

$k=3,$

$$J_p = \frac{\partial(p)}{\partial(\theta_1, \theta_2, \theta_3, \theta_4)}$$

$$= \begin{bmatrix} -xS_1C_{23} + yS_1S_{23} - zC_1 - a_3S_1C_{23} - a_2S_1C_2 - a_1S_1 & -xC_1S_{23} - yC_1C_{23} - a_3C_1S_{23} - a_2C_1S_2 & -xC_1S_{23} - yC_1C_{23} - a_3C_1S_{23} \\ xC_1C_{23} - yC_1S_{23} - zS_1 + a_3C_1C_{23} + a_2C_1C_2 + a_1C_1 & -xS_1S_{23} - yS_1C_{23} - a_3S_1S_{23} - a_2S_1S_2 & -xS_1S_{23} - yS_1C_{23} - a_3S_1S_{23} \\ 0 & -xC_{23} + yS_{23} - a_3 * C_{23} - a_2C_2 & -xC_{23} + yS_{23} - a_3 * C_{23} \end{bmatrix}$$

$$k=4, J_p = \frac{\partial(p)}{\partial(\theta_1, \theta_2, \theta_3, \theta_4)} = [J_{p1}, J_{p2}, J_{p3}, J_{p4}], \text{ and}$$

$$J_{p1} = \begin{bmatrix} -xS_1C_{234} + yC_1 + zS_1S_{234} - a_3S_1C_{23} - a_2S_1C_2 - a_1S_1 \\ xC_1C_{234} + yS_1 - zC_1S_{234} + a_3C_1C_{23} + a_2C_1C_2 + a_1C_1 \\ 0 \end{bmatrix},$$

$$J_{p2} = \begin{bmatrix} -xC_1S_{234} - zC_1C_{234} - a_3C_1S_{23} - a_2C_1S_2 \\ -xS_1S_{234} - zS_1C_{234} - a_3S_1S_{23} - a_2S_1S_2 \\ -xC_{234} + zS_{234} - a_3 * C_{23} - a_2C_2 \end{bmatrix},$$

$$J_{p3} = \begin{bmatrix} -xC_1S_{234} - zC_1C_{234} - a_3C_1S_{23} \\ -xS_1S_{234} - zS_1C_{234} - a_3S_1S_{23} \\ -xC_{234} + zS_{234} - a_3 * C_{23} \end{bmatrix},$$

$$J_{p4} = \begin{bmatrix} -xC_1S_{234} - zC_1C_{234} \\ -xS_1S_{234} - zS_1C_{234} \\ -xC_{234} + zS_{234} \end{bmatrix};$$

$$k=5, J_p = [J_{p1}, J_{p2}, J_{p3}, J_{p4}, J_{p5}], \text{ and}$$

$$J_{p1} = \begin{bmatrix} x(-S_1C_{234}C_5 + C_1S_5) + y(S_1C_{234}S_5 + C_1C_5) + zS_1S_{234} + d_5S_1S_{234} - a_3S_1C_{23} - a_2S_1C_2 - a_1S_1 \\ x(C_1C_{234}C_5 + S_1S_5) + y(-C_1C_{234}S_5 + S_1C_5) - zC_1S_{234} - d_5C_1S_{234} + a_3C_1C_{23} + a_2C_1C_2 + a_1C_1 \\ 0 \end{bmatrix},$$

$$J_{p2} = \begin{bmatrix} -xC_1S_{234}C_5 + yC_1S_{234}S_5 - zC_1C_{234} - d_5C_1C_{234} - a_3C_1S_{23} - a_2C_1S_2 \\ -xS_1S_{234}C_5 + yS_1S_{234}S_5 - zS_1C_{234} - d_5S_1C_{234} - a_3S_1S_{23} - a_2S_1S_2 \\ -xC_{234}C_5 + yC_{234}S_5 + zS_{234} + d_5S_{234} - a_3 * C_{23} - a_2C_2 \end{bmatrix},$$

$$J_{p3} = \begin{bmatrix} -xC_1S_{234}C_5 + yC_1S_{234}S_5 - zC_1C_{234} - d_5C_1C_{234} - a_3C_1S_{23} \\ -xS_1S_{234}C_5 + yS_1S_{234}S_5 - zS_1C_{234} - d_5S_1C_{234} - a_3S_1S_{23} \\ -xC_{234}C_5 + yC_{234}S_5 + zS_{234} + d_5S_{234} - a_3 * C_{23} \end{bmatrix},$$

$$J_{p4} = \begin{bmatrix} -xC_1S_{234}C_5 + yC_1S_{234}S_5 - zC_1C_{234} - d_5C_1C_{234} \\ -xS_1S_{234}C_5 + yS_1S_{234}S_5 - zS_1C_{234} - d_5S_1C_{234} \\ -xC_{234}C_5 + yC_{234}S_5 + zS_{234} + d_5S_{234} \end{bmatrix},$$

$$J_{p5} = \begin{bmatrix} x(-C_1 C_{234} S_5 + S_1 C_5) + y(-C_1 C_{234} C_5 - S_1 S_5) \\ x(-S_1 C_{234} S_5 - C_1 C_5) + y(-S_1 C_{234} C_5 + C_1 S_5) \\ x S_{234} S_5 + y S_{234} C_5 \end{bmatrix}.$$

where, $c_i = \cos \theta_i$, $s_i = \sin \theta_i$, $c_{ij} = \cos(\theta_i + \theta_j)$, $s_{ij} = \sin(\theta_i + \theta_j)$, $c_{ijk} = \cos(\theta_i + \theta_j + \theta_k)$, $s_{ijk} = \sin(\theta_i + \theta_j + \theta_k)$, with parameters a_i , d_i , θ_i obeying the Denavit-Hartenberg (D-H) convention. The J_p value when $k=5$ and $x=y=z=0$ corresponds to the Jacobian for the end-effector center.

B.3 General inverse Jacobian of the Mitsubishi PA-10 robot

The sub-Jacobian Matrixes of the Mitsubishi PA-10 robot with 7 DOF are given below. The reference point p with local coordinate (x, y, z) may be located at the k -th link, and $k=1, \dots, 7$.

$k=1$,

$$J_p = \begin{bmatrix} -xS_1 - zC_1 \\ xC_1 - zS_1 \\ 0 \end{bmatrix};$$

$k=2$,

$$J_p = \begin{bmatrix} -xS_1C_2 - yC_1 - zS_1S_2 & -xC_1S_2 + zC_1C_2 \\ xC_1C_2 - yS_1 + zC_1S_2 & -xS_1S_2 + zS_1C_2 \\ 0 & -xC_2 - zS_2 \end{bmatrix};$$

$k=3$, $J_p = [J_{p1}, J_{p2}, J_{p3}]$, and

$$J_{p1} = \begin{bmatrix} x(-S_1C_2C_3 - C_1S_3) + yS_1S_2 + z(S_1C_2S_3 - C_1C_3) - d_3 * S_1S_2 \\ x(C_1C_2C_3 - S_1S_3) - yC_1S_2 + z(-C_1C_2S_3 - S_1C_3) + d_3 * C_1S_2 \\ 0 \end{bmatrix},$$

$$J_{p2} = \begin{bmatrix} -xC_1S_2C_3 - yC_1C_2 + zC_1S_2S_3 + d_3 * C_1C_2 \\ -xS_1S_2C_3 - yS_1C_2 + zS_1S_2S_3 + d_3 * S_1C_2 \\ -xC_2C_3 + yS_2 + zC_2S_3 - d_3 * S_2 \end{bmatrix},$$

$$J_{p3} = \begin{bmatrix} x(-C_1C_2S_3 - S_1C_3) + z(-C_1C_2C_3 + S_1S_3) \\ x(-S_1C_2S_3 + C_1C_3) + z(-S_1C_2C_3 - C_1S_3) \\ xS_2S_3 + zS_2C_3 \end{bmatrix};$$

$k=4$, $J_p = [J_{p1}, J_{p2}, J_{p3}, J_{p4}]$, and

$$J_{p1} = \begin{bmatrix} x(-S_1C_2C_3C_4 - C_1S_3C_4 + S_1S_2S_4) + y(S_1C_2S_3 - C_1C_3) + z(-S_1C_2C_3S_4 - C_1S_3S_4 - S_1S_2C_4) - d_3 * S_1S_2 \\ x(C_1C_2C_3C_4 - S_1S_3C_4 - C_1S_2S_4) + y(-C_1C_2S_3 - S_1C_3) + z(C_1C_2C_3S_4 - S_1S_3S_4 + C_1S_2C_4) + d_3 * C_1S_2 \\ 0 \end{bmatrix},$$

$$J_{p2} = \begin{bmatrix} x(-C_1S_2C_3C_4 - C_1C_2S_4) + yC_1S_2S_3 + z(-C_1S_2C_3S_4 + C_1C_2C_4) + d_3 * C_1C_2 \\ x(-S_1S_2C_3C_4 - S_1C_2S_4) + yS_1S_2S_3 + z(-S_1S_2C_3S_4 + S_1C_2C_4) + d_3 * S_1C_2 \\ x(-C_2C_3C_4 + S_2S_4) + yC_2S_3 + z(-C_2C_3S_4 - S_2C_4) - d_3 * S_2 \end{bmatrix},$$

$$J_{p3} = \begin{bmatrix} x(-C_1C_2S_3C_4 - S_1C_3C_4) + y(-C_1C_2C_3 + S_1S_3) + z(-C_1C_2S_3S_4 - S_1C_3S_4) \\ x(-S_1C_2S_3C_4 + C_1C_3C_4) + y(-S_1C_2C_3 - C_1S_3) + z(-S_1C_2S_3S_4 + C_1C_3S_4) \\ xS_2S_3C_4 + yS_2C_3 + zS_2S_3S_4 \end{bmatrix},$$

$$J_{p4} = \begin{bmatrix} x(-C_1C_2C_3S_4 + S_1S_3S_4 - C_1S_2C_4) + z(C_1C_2C_3C_4 - S_1S_3C_4 - C_1S_2S_4) \\ x(-S_1C_2C_3S_4 - C_1S_3S_4 - S_1S_2C_4) + z(S_1C_2C_3C_4 + C_1S_3C_4 - S_1S_2S_4) \\ x(S_2C_3S_4 - C_2C_4) + z(-S_2C_3C_4 - C_2S_4) \end{bmatrix};$$

$k=5$,

$$J_{p5} = \begin{bmatrix} -xS_5 - zC_5 \\ xC_5 - zS_5 \\ 0 \end{bmatrix}$$

$k=6$,

$$J_p = \begin{bmatrix} -xS_5C_6 - yC_5 - zS_5S_6 & -xC_5S_6 + zC_5C_6 \\ xC_5C_6 - yS_5 + zC_5S_6 & -xS_5S_6 + zS_5C_6 \\ 0 & -xC_6 - zS_6 \end{bmatrix};$$

$k=7$,

$$J_p = \begin{bmatrix} x(-S_5C_6C_7 - C_5S_7) + y(S_5C_6S_7 - C_5C_7) - zS_5S_6 - d_7S_5S_6 & -xC_5C_7 + yC_5S_7 + zC_5C_6 + d_7C_5C_6 & x(-C_5C_6S_7 - S_5C_7) + y(-C_5C_6C_7 + S_5S_7) \\ x(C_5C_6C_7 - S_5S_7) + y(-C_5C_6S_7 - S_5C_7) + zC_5S_6 + d_7C_5S_6 & -xS_5S_7 + yS_5S_7 + zS_5C_6 + d_7S_5C_6 & x(-S_5C_6S_7 + C_5C_7) + y(-S_5C_6C_7 - C_5S_7) \\ 0 & -xC_6C_7 + yC_6S_7 - zS_6 - S_6d_7 & xS_6S_7 + yS_6C_7 \end{bmatrix}$$

The Jacobian for the wrist centre is the J_p value when $k=5$ and $x=y=z=0$. The above

J_p matrixes when $k=5, 6$ and 7 are expressed in the frame F_w which is located at the wrist

center and has the same orientation with the 4-th link frame F_4 .

Appendix D - Basic A* searching algorithm

In A* algorithm, two lists are maintained. The first list, named OPEN, consists of all the nodes in the search graph that are generated but not yet expanded. The other list, named CLOSED, holds the nodes in the graph that have been expanded. At each step, the node with best evaluation (i.e., minimal path cost) among the nodes in OPEN list is found out. The basic flow of the A* searching method is summarized as follows.

1. Add the starting node to the OPEN list.
2. Repeat the following:
 - a) Look for the lowest F cost node on the OPEN list. We refer to this as the current node.
 - b) Switch it to the CLOSED list.
 - c) For each of the 8 nodes adjacent to this current node
 - If it is not walkable or if it is on the closed list, ignore it. Otherwise do the following.
 - If it is not on the OPEN list, add it to the OPEN list. Make the current node the parent of this node. Record the F, G, and H costs of the node.
 - If it is on the OPEN list already, check to see if this path to that node is better, using G cost as the measure. A lower G cost means that this is a better path. If so, change the parent of the node to the current node, and recalculate the G and F scores of the node. If the OPEN list is kept sorted by F score, the list may need to resort to account for the change.
 - d) Stop when
 - Add the target to the OPEN list, in which case the path has been found, or
 - Fail to find the target, and the OPEN list is empty. In this case, there is no path.

Save the path. Walking backwards from the target, go from each node to its parent node until the starting is reached.

Appendix E – Class definition of motion planning and motion control

```

class Robot_Planning
{
//.....
//      members
//.....

private:
    int Dof_C;

public:
    bool fileLoaded;
    static bool PosOK;

//collision management
    static CArray<WObst,WObst> wobsts;
    static WObst* testwobst;

    static int rad_sphere;
    static Plane plane[6];           //the 6 planes of the room
    static Cylinder cyl;             //the 1 cylinders of the robot base
    static TVector* ArrayVel;        //holds velocity of balls
    static TVector* OldPos;          //old position of balls
    static int NumBalls;
    static TVector accel;

//obst in workspace
    static WObst** wsta, **wdyn;
    static int nwobst, nsta, ndyn;
    static int* sub_sta, *sub_dyn;

// file pointers to store C-obstacle maps
    char fpath[20],fname[20],fname1[30],fname2[30],
        fname3[30],fname4[30],buf[10],num_file[30];
    ofstream pfile;
    ifstream infile;
    ofstream Num_out;           //file output stream object storing number in each cells
    ifstream Num_in;           //file input stream object storing number in each cells

//robot entity
    mModel **robLink; //[Dof_C+1];
    static mModel *maze, *cylinder;

//path display
    int step_count; //curr_PathCenter;
    List_float3d WPath;           //the trajectory of end-effector
    float** CPathCenter;         //the trajectory of collion-free path in the C-space
    float** CPath;
    int* knots;

```

```

//Robot Parameters
float* d, *a, *alpha; //[Dof_C+1]           //basic dimation
float *sitaN, *sitaT, *sitaInit, *sitaGoal;   //temporary sita value
float *sitaR, *sitanorm;
float **ntrans;           //transform array   under normal sequence of storage
GLdouble **Trans;

//C-obst
int bitwidth,
    *level_C,           //auxiliary variables for non-uniform 2m tree
    level_CMax,         //the maximum value of level_C
    *treeDof,           //no. of effective coordinate at level i in the tree
    **lev_dof,          //storing the effective dof at each level
    *factor_C,          //pow(2, treeDof[i])
    *NCarry_C,          //no. of bits for carry
    *dim_C,             //no. of bytes for a unit at the current level in the tree
    factor_W, *AR;
float **JointRange, WRange[Dof_W][2], *stepC, stepWf[Dof_W], stepWr[Dof_W];
short3d WR;
float3d RBase;          //initial position of robot base in the workspace
List_float3d* verts;
List_short3d* subEntity, *subTot, *trisObst;
List_short3dDir2* cellsinOrder;
List_float3d* cylinderCenter, *mazeCenter;
float T[16], Taccu[16], Ttemp[16], Tinit[16];
double Tiden[16];
int* Nelem;
int Num_Wr, Num_C;
int64 Num_Wf;
float** jrange;
float *minCord, *maxCord;

bool* nodeExist, *nodeExistold;
int ncell, ncellold; //count, countold;
int maxCell;
int* currCells, *currCellsold;
int* sub_effec;
int** fileLen, **fileLenold, **fileLen1, **fileLen2;
int** maxLen, **maxLenold;           //max length
int* fileLev1, *fileLev2, *fileLev, *fileLevold; //recode the used level no. of a single node
int res_l, res_lold;
int* sameInd;           //record the index at the old wbtrees and leaf trees
bool swap;
int** curr_tree_block;   // current block in the tree
int** curr_tree_byte;    // current byte
int** curr_tree_bit;     // current bit, will be used in phase 2
int* lengofblock_tree;   //length of block in the medium variable
int* numofblock_tree;    //maximum blocks possible in level k of each subtree

```

```

unsigned_int8 ***wbtree,      ***wbtreeold, ***wbtree1, *** wbtree2;
unsigned_int8 ***leafree,    ***leafreeold, ***leafree1, ***leafree2;
unsigned_int8 **wb_sta,  **leaf_sta,
                **wb_tota, **leaf_tota,
                **wb_resuTree, **leaf_resuTree;
int** num_grey, *** grey;

//time
double duration;                //variables recording the program running time
clock_t Tstart, stag1, Tstop;

//file
static ofstream runRec, resuData;
static int n_phase; //n_timer

//Swift++
double** matr_T;
//float* dq_i;
//bool linkTriPreped;
static double matr_R[9];        //rotation matrix
static double matr_P[3];        //translation matrix
static SWIFT_Scene* scene;
static int* id_obst, id_maze,*id_link;

//FIS
static FIS* fis;
static double *dataMatrix, **fisMatrix, **outputMatrix;
static int data_col_n, fis_col_n, fis_row_n; //data_row_n[4];

//aStar
int nscale, notfinished, notStarted, found, found0, goalreached, nonexistent;
bhNode* openList;                //1 dimensional array holding ID# of open list items
asNode** astack;                //store all the free nodes under order of 1st appearing
bool* stackInit;
int* nfreenode;
int pathLength;                //stores length of the found path for critter
int pathLocation;              //stores current position along the chosen path for critter
bhNode* pathBank ;
int pathStatus;                //Path reading variables
int Path;
int Find;
int numopenlist;

//BSTree
bsNode** bstree;

//.....
//      functions
//.....

```

```

public:
    Robot();
    Robot(int);
    virtual ~Robot();

//Robot transformation matrix and their inverse
    void Trans_update(double*); //give the initial config
    void Trans_Norm(float* sita);
    void Trans_GL();
    bool Trans_Inverse(float*, float*, float*, float*, int);

//c-obst maps superposition
    void Cobst_Super(int, bool);

//calling the off-line and on-line planning procedures
    void offline(bool);
    void online();

//geometric related functions
    void Geo_LineScan(float3d*,float3d*,List_short3d*);
    void Geo_short3dInsert(short3d*,List_short3dDir2*, short3dDir2*);
    void Geo_Decomps_verts(List_float3d*, List_short3d*);
    void Geo_Decomps_tri(List_float3d*,List_short3d*,List_short3d*);
    void Geo_nodesMove();
    void Geo_nodesCount();
    void Geo_DecompsMulti(bool);
    void Geo_statist();
    void Geo_Through(List_short3dDir2*, List_short3d*); //, bool);
    void Geo_Through2_rob (List_short3dDir2*, List_short3d*, int**);
    void Geo_ThroughOut(List_short3dDir2*, List_short3d*);
    bool checkxyz( short3d&, int&);

//transform between tree cell and coordinates
    void Tree_cord2tree(float*, int, float**, int, int&);
    void Tree_tree2coord(int, int, float*);
    void Tree_tree2node(int, int, float*, float*);
    void Tree_tree2bit(int, int, int*);
    void Tree_bit2tree(int*, int, int&);

//A* searching algorithm
    void ASTAR_ReInitPathFinder();
    int  ASTAR_FindPath (int, bhNode, bhNode);
    int  ASTAR_neibsearch(bhNode*, int);
    void ASTAR_neibcheck(bhNode*, int, int, int, int);

//Binary Search Tree (BST)
    bsNode* BST_inittree(int, int);
    bsNode* BST_insert2tree(bsNode*, int, int);
    void    BST_travtree(bsNode*, bsNode*, int&);
    int     BST_getno(bsNode*);

```



```

void    BST_freetree(struct bsNode*);
bsNode* BST_optitree(bsNode*,int,int);
void    BST_call_optitree(bsNode*);
int     BST_matchtree(bsNode*&, int, int, int);
bsNode* BST_getnode(bsNode *,int);

//manage the obstacles, like allocation their locations, moving them at each planning cylce
void Colli_preInit();
void Colli_Init();
int  Colli_TestIntPlane(const Plane&, const TVector&, const TVector&, double&,
TVector&);
int  Colli_TestIntCyl(const Cylinder& cylinder, const TVector& position, const TVector&
direction, double& lamda, TVector& pNormal, TVector& newposition);
int  Colli_FindBallCol(TVector&, double&, double, int&, int&);
void Colli_idle();

//Fuzzy
void  FIS_init();
float FIS_call(float);

//Spline functions
float Spline_Blend(int,int,int*, float);
void  Spline_Point(int*, int, int, int, float,float**, float*);
void  Spline_Knots(int*,int,int);
void  Spline_Curve(float**, int, int, int*, int, float**, int);

//Swift++ for distance calculation between 3D objects
void Swift_init();
bool Swift_Steps(float* );
};

class Robot_Control
{
public:
    RobControl();
    //DECLARE_DYNCREATE(RobControl)
    virtual ~Robot_Control();

public:

//class members
    BOARDIOADDRESS ioaddr[3];
    unsigned short BADR0,BADR2, BADR20,BADR22; // MF614 I/O Card base address
    float sitaN[4]; //current joint angles
    int countN[4]; //current motor positions in terms of counting
    int irc_edge[4][2];

    SPid PID[N_Motor];

```

```

float zero_n[4];    //limit switch position
float c_count[4];
float sitaMin[4];
float sitaMax[4];
float sitaN0[4];
float DAHome[4][2];
int  DAMin[4];
int  DAPos[4];
int  TrapMax_m[4];
int  TrapMax_p[4];
int  Tolerance[4];

//ofstream PlotPtr;
FILE *Plotptr;
ofstream runRec2;
//time recording
clock_t Tc_start, Tc_stag1, Tc_stop;
int nround;
int ircI4, ircB4;  //backup the 4th joint reading

//Control functions
void  MF614_Encoder_Init();
int   MF614_Encoder_Read(UINT channel);
void  MF614_Homing();
double MF614_ADCon(UINT ADChannel);
void  MF614_DACon(UINT DAChannel, int Outvalue);
int   MF614_Din();
void  MF614_Dout(int Dout);
void  MF614_Timer_Init();
void  MF614_Stepper(bool);  //control output to the stepper motor
void  PID_Init();
double PID_Update(SPid *pid, double error, double position);

int   InitMove(float*);
bool  Con_Init();
void  Con_zeroOut();
double PID_Update(SPid*, double);
double Con_ProfileCheck(int diff, int diff0, int chan);
void  Con_Jog(int*);          //single joint move
int   Con_Move(clock_t, float*, float*); //robot move

};

```

References

- [1] Y.K. Hwang and N. Ahuja, "Gross Motion Planning - A Survey," *ACM Computing Surveys (CSUR)*, Vol. 24, No. 3, pp.219 - 291, 1992.
- [2] "ABB Robot Applications by Industry," The.ABB.Group, <http://www.abb.com/robotics>.
- [3] C.A. Zazrnecki, "Design and Off-line Programming of a Dual Robot Workcell for Garment Manufacture," *Microprocessors and Microsystems*, Vol. 23, No. 4, pp.225-234, 1999.
- [4] J.C. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.
- [5] T. Lozano-Perez, "Spatial Planning: A Configuration Space Approach," *IEEE Transactions on Computers*, Vol. C-32, No. 2, pp.108-120, 1983.
- [6] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Boston, 2005.
- [7] K. Gupta and A.P. Del Pobil, *Practical Motion Planning in Robotics: Current Approaches and Future Directions*. John Wiley & Sons, West Sussex, 1998.
- [8] R. Brooks and T. Lozano-Perez, "A Subdivision Algorithm in Configuration Space for Findpath with Rotation," in *Proceedings of the 8th Joint Conference on Artificial Intelligence (ICAI)*, Karlsruhe, W. Germany, pp.799-806, 1983.
- [9] B. Krogh, "A Generalized Potential Field Approach to Obstacle Avoidance Control," in *International Conference on Robotics Research: The Next Five Years and Beyond*, Bethlehem, PA, pp.1-15, 1984.
- [10] O. Khatib, "Real-time Obstacle Avoidance for Manipulators and Mobile Robots," *International Journal of Robotics Research*, Vol. 5, No. 6, pp.90-98, 1986.
- [11] K. Souccar, J. Coelho, C. Connolly, and R. Grupen, "Harmonic Functions for Path Planning and Control," in *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, edited by K. Gupta and A. P. d. Pobil. pp., John Wiley & Sons Ltd., West Sussex, 1998.
- [12] C.I. Connolly, J.B. Burns, and R. Weiss, "Path Planning Using Laplace's Equation," in *Proceedings of the IEEE International Conference on Robotics and Automation*,

- Cincinnati, OH, pp.2102-2106, 1991.
- [13] J. Barraquand and J.C. Latombe, "Robot Motion Planning: A Distributed Representation Approach," *International Journal of Robotics Research*, Vol. 10, No. 6, pp.628-649, 1991.
 - [14] E. Rimon and D.E. Koditschek, "Exact Robot Navigation using Artificial Potential Functions," *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 5, pp.501-518, 1992.
 - [15] D.E. Koditschek, "Exact Robot Navigation by Means of Potential Functions: Some Topological Considerations," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Raleigh, NC, pp.1-6, 1987.
 - [16] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic Roadmaps for Path Planning in High-dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation*, Vol. 12, No. 4, pp.566--580, 1996.
 - [17] R. Geraerts and M. Overmars, "A Comparative Study of Probabilistic Roadmap Planners," in *Proceedings of the Workshop on Algorithmic Foundations of Robotics (WAFR'02)*, Nice, France, pp.43-57, 2002.
 - [18] N. Amato and Y. Wu, "A Randomized Roadmap Method for Path and Manipulation Planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Minneapolis, USA, pp.113-120, 1996.
 - [19] D. Hsu, L. Kavraki, J.C. Latombe, R. Motwani, and S. Sorkin, "On Finding Narrow Passages with Probabilistic Roadmap Planners," in *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics: the Algorithmic Perspective*, Houston, USA, pp.141-154, 1998.
 - [20] M.H. Overmars, "A Random Approach to Motion Planning," Department of Computer Science, Utrecht University, Utrecht, The Netherlands 1992.
 - [21] J.J. Kuffner and S.M. Lavalle, "RRT-connect: An Efficient Approach to Single-query Path Planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, USA, pp.995-1001, 2000.
 - [22] S.M. Lavalle, *Planning Algorithms*. Cambridge University Press, Cambridge, U.K, 2006.

- [23] L. Overgaard, H. G.Petersen, and J. W.Perram, "Reactive Motion Planning: A Multi-agent Approach," *Applied Artificial Intelligence*, Vol. 10, No. 1, pp.35-51, 1996.
- [24] Roboticonline, "A Robot Palletizer for the Sugar and Flour Industry," <http://www.roboticonline.com/public/articles/index.cfm?cat=207>.
- [25] J.-C. Latombe, "Motion Planning: A Journey of Robots, Molecules, Digital Actors, and Other Artifacts," *International Journal of Robotics Research*, Vol. 18, No. 11, pp.1119-1128, 1999.
- [26] M.H. Overmars, "Recent Developments in Motion Planning," in *Proceedings of the International Conference on Computational Science*, Amsterdam, The Netherlands, pp.3-13, 2002.
- [27] J.P. Van Den Berg and M.H. Overmars, "Roadmap-Based Motion Planning in Dynamic Environment," *IEEE Transactions on Robotics*, Vol. 21, No. 5, pp.885-897, 2005.
- [28] B. Oliver and L.E. Kavraki, "Decomposition-based Motion Planning: A Framework for Real-Time Motion Planning in High-Dimensional Configuration Spaces," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Seoul, Korea, pp.1469-1474, 2001.
- [29] P.C. Chen and Y.K. Hwang, "SANDROS: A Motion Planner with Performance Proportional to Task Difficulty," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, France, pp.2346-2353, 1992.
- [30] P. Fiorini and Z. Shiller, "Robot Motion Planning in Dynamic Environments," in *International Symposium of Robotic Research*, Munich, Germany, pp.237-248, 1995.
- [31] J. Canny and J. Reif, "New Lower Bound Techniques for Robot Motion Planning Problems," in *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, New York, pp.49-60, 1987.
- [32] H.L. Sng, G.S. Gupta, and C.H. Messom, "Strategy for Collaboration in Robot Soccer," in *Proceedings of the IEEE International Workshop on Electronic Design, Test and Applications (DELTA '02)*, Palmerston North, New Zealand, pp.347-351, 2002.
- [33] H. Seraji and A. Howard, "Behavior-based Robot Navigation on Challenging Terrain: A Fuzzy Logic Approach," *IEEE Transactions on Robotics and Automation*, Vol. 18, No. 3, pp.308-321, 2002.

- [34] H. Surmann and L. Peters, "MORIA-A Robot with Fuzzy Controlled Behaviour," *Studies in Fuzziness and Soft Computing*, Vol. 61, pp.343-365, 2001.
- [35] H. Samet and R.E. Webber, "Hierarchical Data Structures and Algorithms for Computer Graphics, Part I," *IEEE Computer Graphics and Applications*, Vol. 8, No. 3, pp.48-68, 1988.
- [36] M. Kallmann and M. Mataric, "Motion Planning Using Dynamic Roadmaps," in *Proceedings of the IEEE International Conference on Robotics and Automation*, New Orleans, LA, pp.4399-4404, 2004.
- [37] P. Leven and S. Hutchinson, "Toward Real-time Path Planning in Changing Environments," in *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, Nice, France, pp.363-376, 2000.
- [38] S. Kambhampati and L.S. Davis, "Multiresolution Path Planning for Mobile Robots," *IEEE Transactions on Robotics and Automation*, Vol. 2, No. 3, pp.135-145, 1986.
- [39] B.J.H. Verwer, "A Multiresolution Work space, Multiresolution Configuration Space Approach to Solve the Path Planning Problem," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Cincinnati, Ohio, pp.2107-2112, 1990.
- [40] B. Faverjon, "Obstacle Avoidance Using an Octree in the Configuration Space of a Manipulator," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Ottawa, Canada, pp.504-512, 1984.
- [41] Y. Kitamura, T. Tanaka, F. Kishino, and M. Yachida, "3-D Path Planning in a Dynamic Environment Using an Octree and an Artificial Potential Field," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pittsburgh, USA, pp.2474, 1995.
- [42] K. Hamada and Y. Hori, "Octree Based Approach to Real-time Collision-free Path Planning for Robot Manipulator," in *Proceedings of the IEEE 4th International Workshop on Advanced Motion Control (AMC)*, Tsu-City, Japan, pp.705 - 710, 1996.
- [43] F. Prepata and M.I. Shamos, *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.
- [44] E.U. Acar, H. Choset, A.A. Rizzi, P. Atkar, and D. Hull, "Morse Decompositions for

- Coverage Tasks," *International Journal of Robotics Research*, Vol. 21, No. 4, pp.331-344, 2002.
- [45] R. Bohlin and L. Kavraki, "Path Planning Using Lazy PRM," in *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, USA, 2000.
- [46] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo, "Choosing Good Distance Metrics and Local Planners for Probabilistic Roadmap Methods," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Leuven, Belgium, 1998.
- [47] L. Kavraki, M.N. Kolountzakis, and J.C. Latombe, "Analysis of Probabilistic Roadmaps for Path Planning," *IEEE Transactions on Robotics and Automation*, Vol. 14, No. 1, pp.166-171, 1998.
- [48] A.M. Ladd and L.E. Kavraki, "Measure Theoretic Analysis of Probabilistic Path Planning," *IEEE Transactions on Robotics and Automation*, Vol. 20, No. 2, pp.229-242, 2004.
- [49] P. Bessiere, J.M. Ahuactzin, E.G. Talbi, and E. Mazer, "The "Ariadne's Clew" Algorithm: Global Planning with Local Method," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Yokohama, Japan, pp.1373-1380, 1993.
- [50] A. Mclean and T. Mazon, "Incremental Roadmaps and Global Path Planning in Evolving Industrial Environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, pp.101-106, 1996.
- [51] P. Bessiere, E. Mazer, and J.M. Ahuactzin, "Planning in Continuous Space with Forbidden Rigeons: the Ariadne's Clew Algorithm," in *Algorithmic Foundations of Robotics(WAFR)*, edited by K. Goldberg, R. Wilson, and D. Halperin. pp. 39-47, A.K. Peters, Wellsley, 1995.
- [52] D. Hsu, J.C. Latombe, and R. Motwani, "Path Planning in Expansive Configurations Spaces," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Albuquerque, New Mexico, USA, pp.2719-2726, 1997.
- [53] R. Kindel, D. Hsu, J.C.Latombe, and S. Rock, "Kinodynamic Motion Planning Amidst Moving Obstacles," in *Proceedings of the IEEE International Conference on Robotics*

- and Automation*, San Francisco, CA, pp.537-543, 2000.
- [54] D. Hsu, R. Kindel, J.-C. Latombe, and S.M. Rock, "Randomized Kinodynamic Motion Planning with Moving Obstacles," *The International Journal of Robotics Research*, Vol. 21, No. 3, pp.233-256, 2002.
 - [55] S.M. Lavalle and J.J. Kuffner, "Rapidly-exploring Random Trees: Progress and Prospects," in *Algorithmic and Computational Robotics: New Directions*, edited by B. R. Donald, K. M. Lynch, and D. Rus. pp. 293-308, A K Peters, Wellesley, 2001.
 - [56] S.M. Lavalle and J.J. Kuffner, "Randomized kinodynamic planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Detroit, USA, pp.473-479, 1999.
 - [57] S.M. Lavalle and J.J. Kuffner, "Randomized Kinodynamic Planning," *International Journal of Robotics Research*, Vol. 20, No. 5, pp.378-400, 2001.
 - [58] K. Althoefer, L.D. Seneviratne, P. Zavlangas, and B. Krekelberg, "Fuzzy Navigation for Robotic Manipulators," *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, Vol. 6, No. 2, pp.179-188, 1998.
 - [59] T.-Y. Li, "On-Line Robot Motion Planning In Dynamic Environments," PhD Dissertation, Stanford University, 1995.
 - [60] K. Fujimura, *Motion Planning in Dynamic Environments*. Springer-Verlag, Berlin, Heidelberg, Springer, 1991.
 - [61] J.H. Reif and M. Sharir, "Motion Planning in the Presence of Moving Obstacles," *Journal of the ACM (JACM)*, Vol. 41, No. 4, pp.764 - 790, 1994.
 - [62] M. Erdmann and T. Lozano-Perez, "On Multiple Moving Objects," in *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, pp.1419 - 1424, 1986.
 - [63] T.Y. Li and J.C. Latombe, "On-Line Manipulation Planning for two Robot Arms in a Dynamic Environment," *International Journal of Robotics Research*, Vol. 16, No. 2, pp.144-167, 1997.
 - [64] T. Fraichard, "Trajectory Planning in A Dynamic Workspace: A 'State-Time' Approach," *Advanced Robotics*, Vol. 13, No. 1, pp.75-94, 1999.
 - [65] K. Kant and S.W. Zucker, "Toward Efficient Trajectory Planning: The Path-velocity

- Decomposition," *International Journal of Robotics Research*, Vol. 5, No. 3, pp.72-89, 1986.
- [66] P. Fiorini and Z. Shiller, "Motion Planning in Dynamic Environments Using the Relative Velocity Paradigm," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta GA, pp.560-566, 1993.
- [67] S. Sundar and Z. Shiller, "Time Optimal Obstacle Avoidance," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Nagoya, Japan, pp.3075-3080, 1995.
- [68] P. Fiorini and Z. Shiller, "Motion Planning in Dynamic Environments Using Velocity Obstacles," *International Journal of Robotics Research*, Vol. 17, No. 7, pp.760--772, 1998.
- [69] F. Large, Z. Shiller, S. Sekhavat, and C. Laugier, "Towards Real-Time Global Motion Planning in A Dynamic Environment Using the NLVO Concept," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Switzerland, pp.607-612, 2002.
- [70] Z. Shiller, F. Large, and S. Sekhavat, "Motion Planning in Dynamic Environments: Obstacles Moving Along Arbitrary Trajectories," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Seoul, Korea, pp.3716-3721, 2001.
- [71] G.I. Birbilis and N.A. Aspragathos, "Multi-agent Manipulator Control and Moving Obstacle Avoidance," in *The 9th International Symposium on Advances in Robot Kinematics*, Sestri Levante, Italy, 2004.
- [72] M. Mediavilla, J.L. Gonzalez, J.C. Fraile, and J.R. Peran, "Reactive Approach to On-line Path planning for Robot Manipulators in Dynamic Environments," *Robotica*, Vol. 20, No. 4, pp.375-384, 2002.
- [73] L. Overgaard, H. G.Petersen, and J. W.Perram, "A General Algorithm for Dynamic Control of Multilink Robots,," *International Journal of Robotics Research*, Vol. 14, No. 3, pp.281-294, 1995.
- [74] T.Y. Li and Y.C. Shie, "An Incremental Learning Approach to Motion Planning with Roadmap Management," in *Proceedings of the IEEE International Conference on*

- Robotics and Automation*, Washington, USA, pp.3411-3416, 2002.
- [75] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann, "Planning Collision-free Reaching Motions for Interactive Object Manipulation and Grasping," in *Proceedings of Eurographics*, Granada, Spain, pp.313-322, 2003.
- [76] P. Leven and S. Hutchinson, "A Framework for Real-Time Path Planning in Changing Environments," *International Journal of Robotics Research*, Vol. 21, No. 12, pp.999-1030, 2002.
- [77] P. Leven and S. Hutchinson, "Real-Time Path Planning in Changing Environments," in *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, Hanover, New Hampshire, pp.363-376, 2000.
- [78] L. Jaillet and T. Simeon, "A PRM-based Motion Planner for Dynamically Changing Environments," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japon, pp.1606-1611, 2004.
- [79] J. Vannoy and J. Xiao, "Real-Time Adaptive and Trajectory-Optimized Manipulator Motion Planning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004.
- [80] T. Nanayakkara, K. Watanabe, K. Kiguchi, and K. Izumi, "Evolving Obstacle Avoidance Skill of a Seven-link Redundant Manipulator Subjected to End-effector Working Constraints," in *Proceedings of the 6th International Conference on Soft Computing (IIZUKA2000)*, Iizuka, Japan, pp.684-689, 2000.
- [81] I. Foster, "Designing and Building Parallel Programs,"
<http://www-unix.mcs.anl.gov/dbpp>.
- [82] T. Lozano-Perez and P. O'donnell, "Parallel Robot Motion Planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Sacramento, USA, pp.1000-1007, 1991.
- [83] D. Henrich, C. Wurl, and H. Worn, "6 DOF Path Planning in Dynamic Environments - A Parallel On-line Approach," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Leuven, Belgium, pp.330-335, 1998.
- [84] P. Bessiere, "The Ariadne's Clew Algorithm," in *The First Workshop on the Algorithmic Foundations of Robotics*, San Francisco, CA, 1994.

- [85] D. Challou, D. Boley, and M. Gini, "Parallel Search Algorithms for Robot Motion Planning," in *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, edited by K. Gupta and A. Pobil. pp., John Wiley & Sons Ltd, West Sussex, 1998.
- [86] D. Chalou, M. Gini, and V. Kumer, "Parallel Search Algorithms for Robot Motion Planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, pp.24-51, 1993.
- [87] H. Asama, A. Matsumoto, and Y. Ishida, "Design of an Autonomous and Distributed Robot System: ACTRESS," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tskuba, Japan, pp.283-290, 1989.
- [88] J. Liu and J. Wu, *Multi-agent Robotic Systems*. CRC Press, London, 2001.
- [89] T. Fukuda and G. Iritani, "Construction Mechanism of Group Behavior with Cooperation," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pittsburgh, PA, pp.535-542, 1995.
- [90] M.J. Mataric, "Reward Functions for Accelerated Learning," in *Proceedings of the 11th International Conference on Machine Learning*, San Francisco, pp.181-189, 1994.
- [91] P. Bohner and R. Luppen, "Reactive Multi-agent Based Control of Redundant Manipulators," in *Proceedings of the IEEE International Conference on Robotics and Automation*, New Mexico, USA, pp.1067-1072, 1997.
- [92] D. Gracanin, K.P. Valavanis, N.C. Tsourveloudis, and M. Matijasevic, "Virtual Environment Based Navigation and Control of Underwater Vehicles," *IEEE Robotics and Automation Magazine*, Vol. 6, No. 2, pp.52-63, 1999.
- [93] N.C. Tsourveloudis, K. P. Valavanis, and T. Hebert, "Autonomous Vehicle Navigation Utilizing Electrostatic Potential Fields and Fuzzy Logic," *IEEE Transactions on Robotics and Automation*, Vol. 17, No. 4, pp.490-497, 2001.
- [94] G. Antonelli, S. Chiaverini, R. Finotello, and R. Schiavon, "Real-time Path Planning and Obstacle Avoidance for RAIS: An Autonomous Underwater Vehicle," *IEEE Journal of Oceanic Engineering*, Vol. 26, No. 2, pp.216-227, 2001.
- [95] P.G. Zavlangas and S.G. Tzafestas, "Industrial Robot Navigation and Obstacle Avoidance Employing Fuzzy Logic," *Journal of Intelligent and Robotic Systems*, Vol.

- 27, No. 6, pp.85-97, 2000.
- [96] K. Althoefer, B. Krekelberg, D. Husmeier, and L. Seneviratne, "Reinforcement Learning in A Rule-based Navigator for Robotic Manipulators," *Neurocomputing*, Vol. 37, No. 1, pp.51-70, 2001.
 - [97] G.V.S. Raju, J. Zhou, and R.A. Kisner, "Hierarchical Fuzzy Control," *International Journal of Control*, Vol. 54, No. 5, pp.1201-1216, 1991.
 - [98] E. Tunstel, M.A.A.D. Oliveira, and S. Berman, "Fuzzy Behavior Hierarchies for Multi-robot Control," *International Journal of Intelligent Systems*, Vol. 17, No. 5, pp.449-470, 2002.
 - [99] E.W. Tunstel, "Fuzzy-behavior Synthesis, Coordination, and Evolution in An Adaptive Behavior Hierarchy," in *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, edited by A. Saffiotti and D. Driankov. pp. 205-234, Springer, Heidelberg, New York, 2000.
 - [100] J. Yen and R. Langari, *Fuzzy Logic, Intelligence, Control and Information*. Prentice Hall, New Jersey, 1999.
 - [101] "Source Code of Fuzzy Logic Module in C Language," 6.5 ed: Mathworks Comp., 2002.
 - [102] S.A. Ehmann, "SWIFT++: Speedy Walking via Improved Feature Testing for Non-Convex Objects," <http://www.cs.unc.edu/~geom/SWIFT++/>.
 - [103] J.R. Llata, E.G. Sarabia, J. Arce, and J.P. Oria, "Fuzzy Controller for Obstacle Avoidance in Robotic Manipulators Using Ultrasonic Sensors," in *The 5th International Workshop on Advanced Motion Control(AMC'98)*, Coimbra, Portugal, pp.647 - 652, 1998.
 - [104] D. Grumm, "Optimizing Functions Using ASCFIT," <http://www.adass.org/adass/proceedings/adass98/grummdm/>.
 - [105] W.L. Xu, S.K. Tso, and Z.K. Lu, "A Virtual Target Approach for Resolving the Limit Cycle Problem in Navigation of A Fuzzy Behaviour-based Mobile Robot," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Victoria, Canada, pp.44-49, 1998.
 - [106] C.I. Connolly, "Applications of Harmonic Functions to Robotics," in *Proceedings of the*

- 1992 International Symposium on Intelligent Control*, Glasgow, Scotland, pp.498-502, 1992.
- [107] W. Wei, J.B. Mbede, and Q. Zhang, "Fuzzy Sensor-based Motion Control Among Dynamic Obstacles for Intelligent Rigid-link Electrically Driven Arm Manipulators," *Journal of Intelligent and Robotic Systems*, Vol. 30, No. 1, pp.49-71, 2001.
- [108] P. Isto, "Path Planning by Multiheuristic Search via Subgoals," in *Proceedings of the 27th International Symposium on Industrial Robots*, CEU, Milan, pp.712-726, 1996.
- [109] C. Qin and D. Henrich, "Path Planning for Industrial Robot Arms," in *Proceedings of the International Symposium on Intelligent Robotic Systems(SIRS'96)*, Lisbon, Portugal, pp.65-72, 1996.
- [110] D. Henrich, H. Wörn, and C. Wurll, "On-line Path Planning by Heuristic Hierarchical Search," in *the 24th Annual Conference of the IEEE Industrial Electronics Society (IECON'98)*, Aachen, Germany, pp.2239-2244, 1998.
- [111] B. Paden, A. Mees, and M. Fisher, "Path Planning using a Jacobian-based Freespace Generation Algorithm," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Scottsdale, Arizona, USA, pp.1732 - 1737, 1989.
- [112] V. Hayward, "Fast Collision Detection Scheme by Recursive Decomposition of a Manipulator Workspace," in *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, USA, pp.1044-1049, 1986.
- [113] A. Smith, Y. Kitamura, and F. Kishino, "Efficient Algorithms for Octree Motion," in *IAPR Workshop on Machine Vision Applications*, Kawasaki, Japan, pp.172-177, 1994.
- [114] H. Samet and A. Kochut, "Octree Approximation and Compression Methods," in *Proceedings of the 1st International Symposium on 3D data processing visualization and Transmission(3DPVT'02)*, Padova, Italy, pp.460 - 469, 2002.
- [115] H. Samet, *Applications of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.
- [116] M. Williams and D.I. Jones, "A Rapid Method for Planning Paths in Three Dimensions for a Small Aerial Robot," *Robotica*, Vol. 19, No. 2, pp.125-135, 2001.
- [117] I. Gargantini, "An Efficient Way to Represent Quadtrees," *Communications of the ACM*, Vol. 25, No. 12, pp.905-910, 1982.

- [118] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 1990.
- [119] P.C. Chen and Y.K. Hwang, "SANDROS: A Dynamic Graph Search Algorithm for Motion Planning," *IEEE Transactions on Robotics and Automation*, Vol. 14, No. 3, 1998.
- [120] T. Lozano-Perez, "A Simple Motion Planning Algorithm for General Robot Manipulators," *IEEE Transactions on Robotics and Automation*, Vol. 3, No. 3, pp.224-238, 1987.
- [121] B. Donald, "A Search Algorithm for Motion Planning with Six Degrees of Freedom," *Artificial Intelligence*, Vol. 31, No. 1, pp.295-353, 1987.
- [122] A. Maciejewski and J. Fox, "Path Planning and the Topology of Configuration Space," *IEEE Transactions on Robotics and Automation*, Vol. 9, No. 4, pp.444-456, 1993.
- [123] C. Zhao, M. Farooq, and M. Bayoumi, "Analytical Solution for Configuration Space Obstacle Computation and Representation," in *The 21st International Conference on Industrial Electronics, Control, and Instrumentation (IECON)*, Orlando, Florida, USA, pp.1278 - 1283, 1995.
- [124] W.S. Newman and M.S. Branicky, "Real-time Configuration Space Transforms for Obstacle Avoidance," *International Journal of Robotics Research*, Vol. 10, No. 6, pp.650-667, 1991.
- [125] L. Kavraki, "Computation of Configuration Space Obstacles Using the Fast Fourier Transform," *IEEE Transactions on Robotics and Automation*, Vol. 11, No. 3, pp.408-413, 1995.
- [126] B. Curto and V. Moreno, "Mathematical Formalism for the Fast Evaluation of the Configuration Space," in *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Albuquerque, USA, pp.194-199, 1997.
- [127] D. Henrich, C. Wurl, and H. Worn, "On-line Path Planning with Optimal C-space Discretization," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Victoria, Canada, 1998.
- [128] E. Ralli and G. Hirzinger, "A Global and Resolution Complete Path Planner for Up to

- 6DOF Robot Manipulators," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Minnesota, USA, pp.3295-3302, 1996.
- [129] A. Amini, J.M. Jagadeesh, and B.J. McGinley, "Three Dimensional Thinning of Octree Encoded Objects," in *Proceedings of the 15th Southern Biomedical Engineering Conference*, Dayton, Ohio, pp.293 - 296, 1996.
- [130] "Binary Heap," Wikipedia(the free encyclopedia),
http://en.wikipedia.org/wiki/Binary_heap.
- [131] P. Lester, "Using Binary Heaps in A* Pathfinding,"
<http://www.policyalmanac.org/games/binaryHeaps.htm>, 2003.
- [132] R. Bohlin, "Path Planning in Practice: Lazy Evaluation on a Multi-resolution Grid," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Maui, Hawaii, USA, pp.49-54, 2001.
- [133] H. Prautzsch, W. Boehm, and M. Paluszny, *Bezier and B-spline Techniques*. Springer, Berlin, New York, 2002.
- [134] Inquirer, "Flash Disk," <http://www.theinquirer.net/?article=26140>, 2006.
- [135] "The Manual of ESHED SCORBOT ER4pc robot," ESHED Company, 1997.
- [136] L.-W. Tsai, *Robot Analysis: The Mechanics of Serial and Parallel Manipulators*. John Wiley & Sons Inc., New York, 1999.
- [137] E.W. Weisstein, "Moore-Penrose Matrix Inverse," MathWorld - A Wolfram Web Resource, <http://mathworld.wolfram.com/Moore-PenroseMatrixInverse.html>.
- [138] S.R. Buss and J.-S. Kim, "Selectively Damped Least Squares for Inverse Kinematics," *Journal of Graphics Tools*, Vol. 10, No. 3, pp.37-49, 2005.
- [139] W. Khalil and E. Dombre, *Modeling, Identification & Control of Robots*. Kogan Page Science, London, Sterling, 2004.

Publications

- [1] X. J. Wu, J. Tang, and K. H. Heng, "On the Construction of Discretized Configuration Space of Manipulators," *Robotica*, Vol. 25, No. 1, pp.1-11, 2007.
- [2] X. J. Wu, J. Tang, Q. Li and K. H. Heng, " Development of a Configuration Space Motion Planner for Robot in Dynamic Environment," *Robotics and Computer-Integrated Manufacturing*, accepted, 2007.
- [3] X. J. Wu, J. Tang, Q. Li and K. H. Heng, "A Vector-Format Fuzzy Logic Approach for On-line Robot Motion Planning in 3D Space And Its Application to Underwater Robotics Vehicle", *Robotica*, Vol. 25, No. 3, pp.325-339, 2007.
- [4] X. J. Wu, Q. Li, and K. H. Heng, "Development of a general manipulator path planner using fuzzy reasoning," *Industrial Robot*, vol. 32, 2005.
- [5] X. J. Wu, Q. Li, and K. H. Heng, "A New Algorithm for Construction of Discretized Configuration Space Obstacle and Collision Detection of Manipulators," in *12th International Conference on Advanced Robotics (ICAR2005)*, Seattle, USA, 2005.
- [6] X. J. Wu, Q. Li, and K. H. Heng, "Development of a Vector-Based Fuzzy Logic Approach for Motion Planning," in *IEEE Conference on Cybernetics and Intelligent Systems*, Singapore, 2004.
- [7] Q. Li, X. J. Wu, and K. H. Heng, "General Manipulator Path Planner: A Multi-Agent Based Fuzzy Reasoning Approach," in *Fourth International ICSC symposium on Engineering of Intelligent Systems (EIS 2004)*, Portugal, 2004.