

BOOK NO: 1818836



Bound by
Abbey
Bookbinding Co.

116 Cathays Terrace, Cardiff CF24 4HY
South Wales, U.K. Tel: (029) 2039 5882
www.bookbindersuk.com

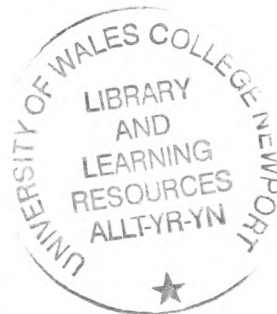
**NOT TO BE
TAKEN AWAY**



A reduced visibility graph approach for motion planning of autonomously guided vehicles

Thesis Submitted to the University of Wales for the Degree of

Doctor of Philosophy



By

Anastasios Diamantopoulos, BSc, MSc
Mechatronics Research Centre
University of Wales College, Newport
June 2001

*I dedicate this thesis
to my mother Alexandra,
whose love and support
I could not do without*

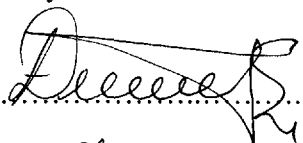
Plan, v.t. To bother about the best method of accomplishing an accidental result.

The Devil's Dictionary, Ambrose Pierce

Declarations

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

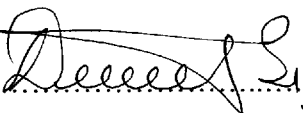
Signed  (candidate)

Date 18 June 2001

STATEMENT 1

This thesis is a result of my own investigations, except where otherwise is stated.

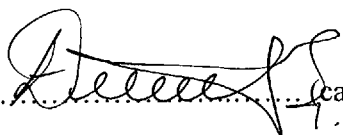
Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed  (candidate)

Date 18 June 2001

STATEMENT 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed  (candidate)

Date 18 June 2001

Acknowledgements

I would like to express my sincere thanks to my former supervisor, Professor Geoff Roberts for his invaluable advice, guidance, motivation and most of all for his support throughout the course of this research. I am also grateful to my second supervisor Dr David Harwood for his advice and support.

I would like to thank to Miss Claire Chambers for her time in reading the thesis and her genuine help and advice with regards to the language and grammar usage.

Thanks are due to all my colleagues in the Mechatronics Research Centre, especially to Mr Alexandros Mouzakis, Dr Ioannis Akkizidis, Dr Antonio Zirilli, Miss Anne Spengler and Mr Eric Llewellyn, for their advice, help and friendship. I am also grateful to Dr Neil Rothwell Hughes from the Engineering Department of the University of Wales College, Newport for his kind help and advice upon several matters concerning my thesis

I would like to thank the staff of the Allt-yr-yn Campus library, who were always happy to help me find all the papers and books I needed during my research. I would especially like to thank Ms Dawne Leatherdale, Mrs Bronwen Stone, Ms Jenifer Coleman and Mr Nick Roberts for their help and support.

Thanks are due to the University of Wales College, Newport for the bursary that was granted to me during the three years of my research. Also I would like to thank the Greek Orthodox Charity Organisation in London for the kind provision of financial support during the difficult period I encountered, whilst writing up my thesis.

Special thanks are due to my family, my sister Helena, my mother Alexandra and my father George, for their endless support, love and encouragement during my studies. I would also like to express my deep appreciation to my grandfather Mr Anastasios Katsandris for his continued support and his kind interest about the progress of my research.

Summary

This thesis is concerned with the robots' motion planning problem. In particular it is focused on the path planning and motion planning for Autonomously Guided Vehicles (AGVs) in well-structured, two-dimensional static and dynamic environments.

Two algorithms are proposed for solving the aforementioned problems. The first algorithm establishes the shortest collision-semi-free path for an AGV from its start point to its goal point, in a two-dimensional static environment populated by simple polygonal obstacles. This algorithm constructs and searches a reduced visibility graph, within the AGV's configuration space, using heuristic information about the problem domain.

The second algorithm establishes the time-minimal collision-semi-free motion for an AGV, from its start point to its goal point, in a two-dimensional dynamic environment populated by simple polygonal obstacles. This algorithm considers the AGV's space-time configuration space, thus reducing the dynamic motion planning problem to the static path planning problem. A reduced visibility graph is then constructed and searched using information about the problem domain, in the AGV's space-time configuration space in order to establish the time-minimal motion between the AGV's start and goal configurations.

The latter algorithm is extended to solve more complicated instances of the dynamic motion planning problem, where the AGV's environment is populated by obstacles, which change their size as well as their position over time and obstacles, which have piecewise linear motion.

The proposed algorithms can be used to efficiently and safely navigate AGVs in well-structured environments. For example, for the navigation of an AGV, in industrial environments, where it operates as part of the manufacturing process or in chemical and nuclear plants, where the hostile environment is inaccessible to humans.

The main contributions in this thesis are, the systematic study of the V*GRAPH algorithm and identification of its methodic and algorithmic deficiencies; recommendation of corrections and further improvements on the V*GRAPH algorithm, which in turn lead to the proposition of the V*MECHA algorithm for robot path planning; proposition of the D*MECHA algorithm for motion planning in dynamic environments; extension to the D*MECHA algorithm to solve more complicated instances of the dynamic robot motion planning problem; discussion of formal proofs of the proposed algorithms' correctness and optimality and critical comparisons with existing similar algorithms for solving the motion planning problem.

Table of Contents

Declarations.....	i
Acknowledgements.....	ii
Summary.....	iii
Table of Contents.....	iv
List of Figures.....	ix
List of Tables.....	xvi
Notations.....	xviii

Chapter 1. Introduction and Overview of the Thesis..... 1-1

1.1 Introduction.....	1-1
1.2 The Importance of Motion Planning in Robotics.....	1-3
1.3 Aim and Objectives of the Research.....	1-8
1.4 Methodology.....	1-9
1.5 Overview of the Thesis.....	1-12
1.6 Discussion.....	1-14
1.7 References.....	1-16

Chapter 2. Preliminaries..... 2-1

2.1 Introduction.....	2-1
2.2 Algorithmic Research.....	2-2
2.2.1 Asymptotic Analysis.....	2-4
2.2.2 Time and Space.....	2-8

2.2.3	Hardness of the Problems.....	2-8
2.3	Computational Geometry and Topology.....	2-13
2.3.1	Higher-Dimensional Spaces.....	2-15
2.4	Configuration Space and Degrees of Freedom.....	2-16
2.4.1	Configuration Space of a Robot.....	2-18
2.4.2	Mapping the Obstacles in the Configuration Space.....	2-19
2.4.3	Path in the Configuration Space.....	2-19
2.4.4	Computing the Obstacles' Configuration Space.....	2-20
2.5	Discussion.....	2-23
2.6	References.....	2-24
 Chapter 3. Robot Motion Planning – Literature Survey.....		3-1
3.1	Introduction.....	3-1
3.2	Roadmap Approaches.....	3-3
3.2.1	Visibility Graph.....	3-3
3.2.2	Generalised Voronoi Diagram.....	3-9
3.2.3	Freeway Method.....	3-14
3.2.4	Silhouette Method.....	3-20
3.2.5	Probabilistic Roadmaps (PRM).....	3-27
3.3	Cell Decomposition Approaches.....	3-30
3.3.1	Exact Cell Decomposition.....	3-31
3.3.2	Approximate Cell Decomposition.....	3-41
3.4	Potential Field Approach.....	3-48
3.4.1	Randomised Path Planning.....	3-55
3.5	Discussion.....	3-56
3.6	References.....	3-58

Chapter 4. The V*MECHA Algorithm for Path Planning of an AGV.....	4-1
4.1 Introduction.....	4-1
4.2 Related Work.....	4-4
4.3 A Description of the V*GRAPH Algorithm.....	4-7
4.3.1 The A* Algorithm.....	4-7
4.3.2 The V*GRAPH Algorithm.....	4-11
4.4 Conn <i>et al</i> (1997)'s Counterexample on the V*GRAPH Algorithm.....	4-14
4.4.1 The Bug in the V*GRAPH Algorithm.....	4-17
4.5 Identification of a New Deficiency on the V*GRAPH Algorithm.....	4-18
4.6 Proposition of the V*MECHA Algorithm for Path Planning.....	4-20
4.6.1 The V*MECHA Algorithm.....	4-32
4.6.2 V*MECHA's Subroutines.....	4-36
4.7 Testing V*MECHA using Conn <i>et al</i> (1997)'s Counterexample.....	4-43
4.8 The Admissibility of the V*MECHA Algorithm.....	4-47
4.9 The Optimality of the V*MECHA Algorithm.....	4-50
4.10 The Optimality of the Path Produced by the V*MECHA Algorithm.....	4-54
4.11 Time and Space Complexities of the V*MECHA Algorithm.....	4-55
4.12 Comparing the V*MECHA Algorithm to other Visibility Graph Approaches.....	4-57
4.13 Discussion.....	4-62
4.14 References.....	4-63
 Chapter 5. The D*MECHA Algorithm for Motion Planning of an AGV in Dynamic Environments.....	 5-1
5.1 Introduction.....	5-1
5.2 Related Work.....	5-4
5.3 The Space-Time Configuration Space.....	5-7
5.4 'Reachability' and Visibility.....	5-11
5.5 Proposition of the D*MECHA Algorithm.....	5-16
5.5.1 The D*MECHA Algorithm.....	5-24

5.5.2	The D*MECHA's Subroutines.....	5-29
5.6	A Simple Motion Planning Problem.....	5-32
5.7	The Admissibility and the Optimality of the D*MECHA Algorithm.....	5-39
5.8	The Optimality of the Motion Produced by the D*MECHA Algorithm.....	5-40
5.9	Time and Space Complexities of the D*MECHA Algorithm.....	5-48
5.10	Some Interesting Properties of the Motion's Time Minimality.....	5-49
5.11	Comparing the D*MECHA Algorithm to other Approaches.....	5-51
5.12	Discussion.....	5-53
5.13	References.....	5-55
 Chapter 6. Extensions on the D*MECHA Algorithm.....		6-1
6.1	Introduction.....	6-1
6.2	Environments with Shrinking and Expanding Obstacles.....	6-4
6.2.1	Detailed Description of the Shrinking and Expanding Obstacles.....	6-6
6.2.2	Construction of the Space-Time Configuration Space.....	6-8
6.2.3	Applicability of the D*MECHA Algorithm.....	6-9
6.2.4	Sensitivity Analysis of the Admissibility and Optimality Properties of the D*MECHA Algorithm.....	6-10
6.3	Environments Containing Piecewise Linearly Moving Obstacles.....	6-11
6.3.1	Space-Time Configuration Space with Piecewise Linear Obstacles.....	6-12
6.3.2	Applicability of the D*MECHA Algorithm.....	6-14
6.3.3	Extension on the D*MECHA Algorithm.....	6-23
6.3.4	Analysis of the Time and Space Complexities.....	6-29
6.3.5	The Optimality of the Motion Produced by the D*MECHA Algorithm..	6-30
6.3.6	Comparison with other Approaches.....	6-31
6.4	Discussion.....	6-38
6.5	References.....	6-40

Chapter 7. Conclusions and Future Work.....	7-1
7.1 Introduction.....	7-1
7.2 Review of the Thesis.....	7-2
7.3 Discussions and Conclusions of the Research.....	7-5
7.4 The Main Contributions of the Thesis.....	7-9
7.5 Suggestions for Future Work.....	7-13
7.6 References.....	7-15
Appendix A. Computation of the C-Obstacles using Minkowski Sums.....	A-1
A.1 Configuration Space of a Robot.....	A-2
A.2 Minkowski Sums.....	A-5
Appendix B. The A* Algorithm.....	B-1
B.1 Graph Searching.....	B-2
B.2 A General Graph Search Approach.....	B-2
B.3 Heuristic Information.....	B-4
B.4 The A* Search Algorithm.....	B-5
B.5 The Admissibility of the A* Algorithm.....	B-8
B.6 The Optimality of the A* Algorithm.....	B-11
Appendix C. Publications.....	C-1

List of Figures

Chapter 1

Figure 1.1	An autonomous robotic system.....	1-5
-------------------	-----------------------------------	-----

Chapter 2

Figure 2.1	Illustration of the TOWERS OF HANOI problem.....	2-9
Figure 2.2	Relation between the problems, whose solution requires exponential time algorithm.....	2-12
Figure 2.3	Illustration of different types of robots. On the left there is a polygonal robot with three degrees of freedom and on the right there is a robotic manipulator with two degrees of freedom.....	2-16
Figure 2.4	The shaded and grey areas constitute the grown obstacle or the obstacle's configuration space CP.....	2-22

Chapter 3

Figure 3.1	Visibility graph with polygonal obstacles.....	3-4
Figure 3.2	Edges of the Tangent Graph are only the cotangents of any pair of the scene's C-Obstacles.....	3-8
Figure 3.3	Voronoi diagram for eight sites in the plane Method.....	3-10
Figure 3.4	The Generalised Voronoi Diagram for a set of convex obstacles in the plane bounded by an external polygonal obstacle Method.....	3-12
Figure 3.5	Illustration of the retraction of q_{start} and q_{goal} on the $GVD(C_{free})$	3-13

Figure 3.6	A two dimensional straight linear generalised cylinder created by the environment's obstacles P_1 and P_2	3-15
Figure 3.7	Truncation of non-free slices of a generalised cone.....	3-17
Figure 3.8	Five freeways generated by the environments boundary and the environment's obstacles.....	3-18
Figure 3.9	Illustration of S , which is a compact sub-set of R^m	3-21
Figure 3.10	The slice $P_c \cap S$ of S when the P_c is at position $x = c$	3-22
Figure 3.11	The two extremal points of the slice $P_c \cap S$ in y -direction.....	3-23
Figure 3.12	When the $P_c \cap S$ is swept across the x -axis the extremal points form silhouette curves (long-dashed curves).....	3-24
Figure 3.13	At the critical values c_i new extremal appear in the set S	3-25
Figure 3.14	The complete silhouette of set S	3-26
Figure 3.15	The robot's configuration space externally bounded by an obstacle and internally bounded by four obstacles.....	3-32
Figure 3.16	Trapezoidal decomposition of the C_{free}	3-34
Figure 3.17	The connectivity graph G , which represents the adjacency of the cells of Figure 3.16. The bold line is the path from the cell, which contains q_{start} to the cell, which contains q_{goal}	3-35
Figure 3.18	The channel (shaded cells) is a sequence of adjacent cells between the cells which contain q_{start} and q_{goal}	3-35
Figure 3.19	The path between q_{start} and q_{goal}	3-36
Figure 3.20	Two-dimensional configuration space.....	3-42
Figure 3.21	The cells obtained after two successive decompositions of the environment.....	3-43
Figure 3.22	The connectivity graph G , representing the adjacency of the cells of Figure 3.21.....	3-44
Figure 3.23	Cells obtained after the third decompositions of R	3-45
Figure 3.24	The connectivity graph G , representing the adjacency of the cells of Figure 3.23.....	3-46
Figure 3.25	The path (dotted line), between q_{start} and q_{goal}	3-46
Figure 3.26	Illustration of a case where additional vertices should be introduced.....	3-47

Figure 3.27	The summation of the artificial attractive and repulsive potential fields of a two-dimensional environment.....	3-51
Figure 3.28	The parabolic well created by the artificial attractive potential field of the goal point in a two-dimensional environment.....	3-52
Figure 3.29	The artificial repulsive potential field of the obstacles in a two-dimensional environment.....	3-53

Chapter 4

Figure 4.1	2, 4, 5, 7 and 8 are obtuse s-visible vertices. 1, 3, 6, 8 and 9 are the extreme vertices of the s-visible sequences.....	4-11
Figure 4.2	Illustration of the scene of the counter example.....	4-15
Figure 4.3	Illustration of the collision path produced by the V*GRAPH algorithm for the scenario of Figure 4.2.....	4-16
Figure 4.4a	Vertex x is an acute vertex.....	4-19
Figure 4.4b	Vertex x is an obtuse concave vertex.....	4-19
Figure 4.4c	Vertex x is an obtuse non-concave vertex.....	4-19
Figure 4.5	Illustration of an environment where the V*GRAPH algorithm fails to find a path, due to the fact that it does not consider obstacles' obtuse vertices for the construction of the visibility graph.....	4-19
Figure 4.6	The shortest path bends only at the obstacles' vertices.....	4-22
Figure 4.7	1, 3, 9, 10, 15 and 16 are the extreme vertices of the s-visible sequences.....	4-24
Figure 4.8	The shortest path between s and g though k only visits extreme vertices of the k-visible sequences.....	4-25
Figure 4.9	The super-extremes of the k-visible sequences are the vertices 1 and 9.....	4-27
Figure 4.10	The shortest path never visits concave vertices.....	4-29
Figure 4.11	Illustration of the reduction due to Proposition 4.1 and Proposition 4.2.....	4-31

Figure 4.12	Environment used for the demonstration of the <i>EXTREME_VERTICES</i> subroutine	4-41
Figure 4.13	Illustration of the counterexample's scene.....	4-43
Figure 4.14	The arrows illustrate the semi-free path proposed as a solution to the scenario of Figure 4.13 by the V*MECHA algorithm.....	4-46
Figure 4.15	The environment used in (Liu and Arimoto, 1992) for the comparison of VGRAPH and T-graph algorithms.....	4-59
Figure 4.16a	Illustration of the visibility graph constructed by the VGRAPH algorithm for the identification of the shortest path between the AGV's start point s and its goal point g	4-60
Figure 4.16b	Illustration of the visibility graph constructed by the V*MECHA algorithm for the identification of the shortest path between the AGV's start point s and its goal point g	4-60
Figure 4.17a	Illustration of the visibility graph constructed by the T-graph algorithm for the identification of the shortest path between the AGV's start point s and its goal point g	4-61
Figure 4.17b	Illustration of the visibility graph constructed by the V*MECHA algorithm for the identification of the shortest path between the AGV's start point s and its goal point g	4-61

Chapter 5

Figure 5.1	The AGV's workspace W populated by a stationary and a moving obstacles as well as the AGV's start and goal points.....	5-9
Figure 5.2a	The AGV's configuration space C at time t_α	5-9
Figure 5.2b	The AGV's configuration space C at time t_β	5-9
Figure 5.3	The AGV's space-time configuration space CT for the environmen of Figure 5.1.....	5-10
Figure 5.4	All the reachable configurations from p are bounded by the perimeter of the circle CR	5-12

Figure 5.5	All the reachable configurations from p are defined by the surface of the cone CN	5-13
Figure 5.6a	'Reachability' in CT	5-15
Figure 5.6b	'Reachability' in C	5-15
Figure 5.7	The time-minimal motion does not visit non-extreme vertices of the visible sequence.....	5-19
Figure 5.8	The time-minimal motion does not visit concave vertices.....	5-23
Figure 5.9	Treatment of the shaft edges considered for the construction of the RVG_9 that have been visited before.....	5-26
Figure 5.10	Illustration of the problem's scene.....	5-32
Figure 5.11	Illustration of the AGV's space-time configuration space CT	5-34
Figure 5.12a	The black dots indicate all the configurations placed in the tree motion at the first iteration of the algorithm.....	5-36
Figure 5.12b	The black dots indicate all the configurations placed in the tree motion at the second iteration of the algorithm.....	5-36
Figure 5.12c	The black dots indicate all the configurations placed in the tree motion at the third iteration of the algorithm.....	5-37
Figure 5.12d	The black dots indicate all the configurations placed in the tree motion at the forth iteration of the algorithm.....	5-37
Figure 5.13	The arrows illustrate the semi-free motion proposed as a solution to the problem of Figure 5.10 from the D*MECHA algorithm.....	5-38
Figure 5.14	Illustration of the case where q_g is visible from q_2 at time t_2	5-41
Figure 5.15	Illustration of the case where q_g is not visible from q_2 at t_2 but it becomes so while the AGV is moving towards q_g	5-43
Figure 5.16a	Illustration of the case where q_g is not visible from q_2 at t_2 and it does not become so even while the AGV is moving towards q_g	5-44
Figure 5.16b	For the case depicted in Figure 5.16a the AGV will have to pass through another vertex of the CP_1 on its way to q_g	5-44

Chapter 6

Figure 6.1	Illustration of an obstacle, which expands about its internal point O.....	6-6
Figure 6.2a	The AGV's configuration space containing one linearly moving C-Obstacle and one C-Obstacle which shrinks.....	6-9
Figure 6.2b	The AGV's space-time configuration space for the configuration space of Figure 6.2a.....	6-9
Figure 6.3a	The AGV's configuration space C populated by the linearly moving C-Obstacle CP.....	6-13
Figure 6.3b	The AGV's configuration space-time CT for the C-space of Figure 6.3a.....	6-13
Figure 6.4	An example of a C-space, which contains one piecewise linearly moving C-Obstacle.....	6-15
Figure 6.5	Illustration of a stop-motion.....	6-17
Figure 6.6	Vertices 1, 2 and 4 would have been visible and reachable at configurations q_1 , q_2 and q_4 at times t_1 , t_2 and t_4 respectively, if the CP had not changed the direction of its motion when it reached line ℓ_2	6-18
Figure 6.7	Vertices 1 and 4 are visible and reachable from q_s at configurations q_1 and q_4' respectively.....	6-19
Figure 6.8	The time-minimal motion from q_s to q_g , which goes around the topside of the CP.....	6-20
Figure 6.9	A motion from q_s to q_g , through configuration $q_{ass}^{(1,2)}$, which corresponds to an internal point of the CP's edge (1, 2).....	6-22
Figure 6.10	Illustration of an assistant configuration (q_{ass}), which is visible and reachable from configuration q in CT.....	6-24
Figure 6.11	Illustration of the internal assistant configurations.....	6-25
Figure 6.12	Assistant configurations $q_{ass}^{(x)}$ and $q_{ass}^{(y)}$ are on the same edge.....	6-27
Figure 6.13	Illustration of the four motions.....	6-33
Figure 6.14a	Profile of the velocity and the duration of the motion defined by Kant and Zucker's approach.....	6-36
Figure 6.14b	Profile of the velocity and the duration of the motion defined	

	by Erdmann and Lozano-Pérez's approach.....	6-36
Figure 6.14c	Profile of the velocity and the duration of the motion defined by Fujimura's approach.....	6-36
Figure 6.14d	Profile of the velocity and the duration of the motion defined by D*MECHA algorithm.....	6-36
Figure 6.15	Velocity profile of the four motions when used to solve the encountered example.....	6-37

Appendix A

Figure A.1	The placement of a robot can be specified in terms of the co-ordinates of its reference point.....	A-3
Figure A.2	The shaded areas constitute the grown obstacle or the obstacle's configuration space CP_k	A-6
Figure A.3	$-S$ is obtained by reflecting S about its origin.....	A-7
Figure A.4	The extreme point of the $P \oplus R$ in direction \vec{u} is the sum of the extreme point of P and R in direction \vec{u}	A-8

List of Tables

Chapter 4

Table 4.1	The co-ordinates of the obstacle's vertices, the robot's start point s and its goal point t.....	4-14
Table 4.2	Summary of the results of the V*GRAPH algorithm when is applied to the scenario of Figure 4.2. The arrow indicates at which point the algorithm fails.....	4-15
Table 4.3	Summarization of the results of the <i>SUPER_EXTREMES</i> subroutine when is applied is applied to the scene of Figure 4.12...	4-42
Table 4.4	The co-ordinates of the scene's features.....	4-44
Table 4.5	Summary of the results of the V*MECHA algorithm when is applied to the scenario of Figure 4.13.....	4-45

Chapter 5

Table 5.1	Co-ordinates of the AGV's start and goal locations and the obstacles' vertices initial locations of the environment of Figure 5.10.....	5-33
Table 5.2	Summary of the results of the D*MECHA algorithm when is applied to the problem of Figure 5.10.....	5-35

Chapter 6

Table 6.1	The co-ordinates of the C-Obstacle's vertices (at their initial position) and the AGV's start and goal configurations.....	6-15
Table 6.2	Summary of the comparison between the four approaches.....	6-35

Notations

Algorithmic Research

$O(.)$	Upper bound
$\Omega(.)$	Lower bound
$\Theta(.)$	Tight bound
P	Polynomial time problems
NP	Non-deterministic Polynomial Problems
NP-complete	Non-deterministic Polynomial complete Problems
NP-hard	Non-deterministic Polynomial hard Problems
NP-easy	Non-deterministic Polynomial easy Problems
PSPACE	Polynomial-Space Problems
PSPACE-complete	Polynomial-Space complete Problems
PSPACE-hard	Polynomial-Space hard Problems
PSPACE-easy	Polynomial-Space easy Problems

Computational Geometry - Topology

F_R	Cartesian co-ordinate frame attached to R
O_R	Origin of the Cartesian co-ordinate frame attached to R
\mathbb{R}^d	d-dimensional space of reals
E^d	d-dimensional Euclidean space
\overline{pq}	Line segment connecting points p and q
$v(S)$	Voronoi diagram of a set of points (sites) S
$d(p, q)$	Euclidean distance between p and q
$d_{air}(p, q)$	Airline Euclidean distance between points p and q

$d_{\text{edge}}(p, q)$	The length cost of an edge connecting vertices p and q in a graph
$d_{\text{min}}(p, P)$	Minimum distance from point p to a geometric feature P

Operators

\oplus	Minkowski sum operator: $S_1 \oplus S_2 = \{a + b : a \in S_1, b \in S_2\}$
\ominus	Minkowski difference operator: $S_1 \ominus S_2 = \{a - b : a \in S_1, b \in S_2\}$

Robotics

dof	Degrees of freedom
R	Robot R
r	Robot's reference point
P_i	Obstacle i
W	The AGV's workspace
C or C_{space}	Configuration space of the AGV
C_{free}	Free configuration space
CP_i	C-Obstacle i or Obstacle's i configuration space
q_s or q_{start}	AGV's start configuration
q_g or q_{goal}	AGV's goal configuration
$R(q)$	Robot R at configuration q

Sets

\mathbb{N}	Set of natural numbers
\mathbb{R}	Set of real numbers
\emptyset	Empty set
$A \subseteq B$	Set A is a subset of set B
$A \subset B$	Set A is a proper subset of set B

$A \cap B$	Intersection of sets A and B
$A \cup B$	Union of sets A and B
$[p, q]$	Closed interval
(p, q) or $]p, q[$	Open interval
∂S	Boundary of the set S
$\text{int}(S)$	Interior of the set S
\bar{S}	Complement of the set S
$\text{cl}(S)$	Closure of the set S

Miscellaneous

ϑ	Angle
\vec{F}	Force produced by an artificial potential field U
\vec{F}_{attr}	Force produced by an artificial attractive potential field U_{attr}
\vec{F}_{rep}	Force produced by an artificial repulsive potential field U_{rep}
U	Artificial potential field
U_{attr}	Artificial attractive potential field
U_{rep}	Artificial repulsive potential field

1

Introduction and Overview of the Thesis

Though thy beginning was small, yet thy end will be very great

JOB 8:7

1.1 Introduction

Imagine an eight-year-old child is playing with a remote controlled car in the living room of his/her house. The child skilfully navigates the car from one corner of the room to another, avoiding collisions with the room's furniture. What the child has just done is to solve the *gross motion planning problem*, for a vehicle with three degrees of freedom. If the action of the child is more thoroughly analysed, it can be noticed that the child using his/her vision, recognises the objects in the car's environment (the living room), identifies their position and thus is able to build up a geometric model of the environment. This model in turn helps him/her to navigate the car around the

environment. Notice that the child unconsciously solves another significant robotics problem, the vision problem.

Imagine now the scenario where the child is blindfolded and tries to navigate the car, from one corner of the living room to another. The result will not be the same. The navigation now relies upon previous experience the child has about the topology of the room. The child knows the approximate position of the furniture in the room and tries to navigate the car in an exploratory manner. For instance, if the car crashes with an object of the room, the child instinctively backs it up and tries to circumnavigate the object and reach the destination point.

This example demonstrates how easy it is for a human to solve the motion planning problem with little conscious effort, providing that information is available about the environment ahead of planning. It appears that when a human intuitively solves the motion planning problem, in general it generates sub-optimal but safe trajectories. The reason is that the human's ability to move safely in a physical environment or to move other objects within the environment, is a process undertaken instinctively, as generally humans do not calculate all the possible routes and then select the optimal.

To replicate this human ability in a computer programme to navigate a robot in a physical environment is an extremely difficult task. The reason is that there are computational aspects involved, which themselves are very hard to undertake. For example, one of the most challenging tasks is to represent real world objects within the computer and to define geometric operations for them. Another important issue is the

computational complexity, i.e. how to generate effective solutions to the motion planning problem in efficient time. The motion planning problem has received great attention from the computer theoreticians community, thus it is sometimes referred to in the literature as *algorithmic motion planning*.

1.2 The Importance of Motion Planning in Robotics

One of the most important tasks a robot should undertake in order to be autonomous is to plan and execute its own motions in known environments or sense, react and avoid obstacles in unknown environments. There are a significant number of applications, which make use of robotic systems and in particular of mobile robots, ranging from manufacturing to space exploration.

The use of AGVs (Autonomously Guided Vehicles) and articulated robots is an important consideration for the efficiency of an automated manufacturing process. In the machining industry the use of robotic systems has made mass production a human unassisted process with very precise results leading to good quality products. Mobile robots are also used to conduct tasks in hostile environments, which are dangerous for humans, such as nuclear and chemical plants, battlefields, mines and outer-space environments.

Notice that most practical applications in which a mobile robot is involved are critical, in the sense that they have to meet safety and economic aspects. For example, it would be economically catastrophic if a mobile robot, which works as part of a manufacturing

process in a multi-million pound production line, collided with other machines and disrupted the process. Even more importantly it would be a disaster if a mobile robot were to harm a human who also co-exists in its environment. The above examples show how important it is for a robot to move safely in its environment and carry out tasks in an optimal way. Therefore, algorithms, which plan the motion of a robot in a physical environment, are of significant importance and they are called *motion planners* or *path planners*. Note however that path planning is not the only ingredient for the guidance of a robotic system. If the guidance procedure of a robotic system is analysed it can be decomposed in the following three tasks:

- The first task is to create a map of the environment (if is not available, i.e. known environment) in which the robot is operating. This task can be achieved by sensing the environment.
- The second task is motion planning. This task establishes safe (collision-free) and cost effective motions for the robot between two query points in its environment.
- The final task is the robot to be driven efficiently along the path established by the previous task. This task controls the robot's motors and actuators to perform the desired motion.

The above process can be schematically represented as illustrated in Figure 1.1.

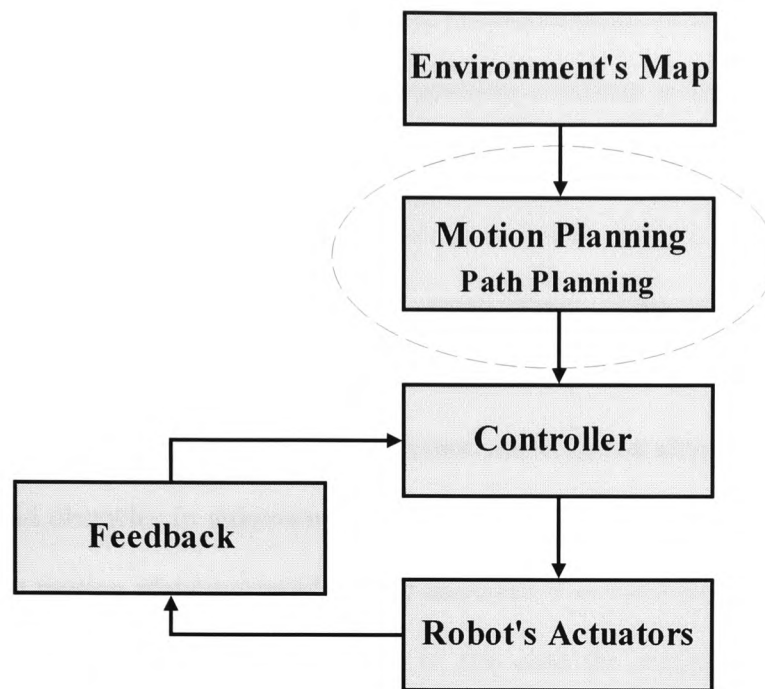


Figure 1.1 An autonomous robotic system.

This thesis describes approaches to provide effective solutions to the second task involved in a robotic system's guidance, i.e. the motion planning task.

The earliest method for programming a robot to undertake a motion was to guide the robot through the motion by specifying a sequence of desired configurations. These configurations would be the robots initial and goal positions and additional intermediate configurations. This method of robot programming is called *teaching by showing* or *guiding* (Lozano-Pérez and Wesley, 1983). Guiding was a common approach for programming robots and is still used today for programming industrial robots because of its simplicity and the relatively low level of operator skill. Robotic systems programmed by guiding perform repetitive tasks such as welding, painting and palletisation well. However, their abilities are limited due to the fact that every time a

different task is assigned to the robot a training procedure needs to be undertaken. Also note that using the guidance method to programme a robotic system, its application domain is very limited because the system cannot cope with moving obstacles.

The ultimate goal of robotics is to create autonomous robots and therefore much research effort is now concentrated on creating autonomously guided robots, which have the ability to plan and execute their own motions in known environments or sense, react and avoid obstacles in unknown environments. Note that this thesis is concerned with the robot motion planning problem and therefore it is considered that the robot's environment is known ahead of planning. In this case the motion planning task is adequately accomplished in an off-line manner. However, when the environment is unknown on-line planning (better described as obstacle avoidance) is required by the system, which relies on information obtained by the robot's sensors.

The importance of the motion planning problem in robotics on the one hand and its hardness¹ on the other, justify the volume of research effort and the large number of proposed approaches dealing with the motion planning problem. A review of robot motion planning is presented in chapter three. Additional surveys can be found in (Yap, 1987), (Schwartz and Sharir, 1988), (Latombe, 1991), (Hwang and Ahuja, 1992) and (Wager, 2000).

It has been shown that complete motion planning approaches, which guarantee to find a solution to the problem if one exists, are computationally so expensive that they limit

¹ In the computer theoreticians' community a problem is commonly characterized as hard when the best available algorithm for solving is very expensive in its running time.

the practicality of the approach, (Schwartz and Sharir, 1983), (Canny, 1988). However, heuristic, resolution-complete, and probabilistically resolution-complete approaches have been developed to make the solution of the motion planning problem more pragmatic but at the expense of its completeness. These issues are also discussed in chapter three.

A relaxed specification of the robot motion planning problem can be stated as follows: Given the initial and final states of a robot and the constraints of allowable motions, find a collision-free motion for the robot from its initial state to its final state which satisfies the constraints. This is a very generic specification of the problem, thus the problem has been classified into categories based upon the robot's environment and task domains. For instance, when the environment of the robot is static and the only constraints on the robot are its kinematics and that it should not collide with the environment's obstacles, it suffices to define collision-free paths for the robot to follow taking into account its kinematics. This problem is referred to in the literature as the *path planning problem*. When the dynamics of the robot are taken into consideration such as linear and angular velocities this problem is referred to as the *trajectory planning problem*. Note that path planning problem is subset of trajectory planning. When the environment of the robot contains only stationary obstacles it is referred to as the *static motion planning problem* while when the environment contains moving obstacles it is referred to as the *dynamic motion planning problem*². More details about

² In some textbooks the static motion planning and dynamic motion planning problem are referred to as time-invariant and time varying motion planning problems respectively.

the classification of the robot motion planning problems can be found in (Latombe, 1991) and (Hwang and Ahuja, 1992).

In this thesis two specific instances of the motion planning problem are considered. The first is the path planning problem for an AGV in a two-dimensional static known environment. The second is the motion planning of an AGV in a two-dimensional dynamic known environment.

1.3 Aim and Objectives of the Research

The aim of the research is to develop algorithms for solving effectively and efficiently the path planning problem for an AGV in two-dimensional static known environments and the motion planning problem for an AGV in two-dimensional dynamic known environments.

The objectives of the research are to:

- Conduct a literature survey on approaches and algorithms that have been proposed for solving robot motion planning problem.
- Study and critically review existing robot motion planning approaches in order to identify major problems and recommend ways to overcome these problems.
- Propose possible improvements to existing approaches for solving the path planning problem for an AGV.
- Propose an approach for solving the motion planning problem for an AGV in dynamic environments.

- Investigate the correctness and the optimality of the proposed approaches.
- Test the proposed approaches experimentally through examples.
- Critically evaluate the proposed approaches and compare them with existing approaches.

1.4 Methodology

As was mentioned in section 1.2 the first problem considered is the path planning problem for an AGV in a two-dimensional static environment. More formally the problem is posed as follows:

Let the AGV R be a simple polygonal free-flying object operating in a two-dimensional Euclidean workspace W , populated by a finite number of simple polygonal obstacles, the AGV's start point s and its goal point g . The problem is to find a collision-free path for the AGV R , given that the positions of the obstacles and the AGV's start and goal positions are known ahead of planning or to report failure if such path does not exist. This is called the *basic path planning problem* or the *basic movers' problem* because there are restrictions on the robot and its environment, such as the robot is a rigid body and no kinematic constraints are imposed on it, the obstacles are static and so on. In Latombe (1991), it is reported that even the basic movers' problem is a hard problem to solve and when the robot is an articulated arm it gets even harder.

The approach proposed in the thesis to solve the problem, is based on the visibility graph approach (Lozano-Pérez and Wesley, 1979), see section 3.2.1 for details. The

general idea behind the visibility graph approach is to construct a graph in the robot's configuration space connecting all the vertices of the obstacles' configuration spaces that can be connected by a straight-line edge, such that this edge does not overlap the interior of any obstacle's configuration space. It then searches this graph for a path between the AGV's start and goal locations.

The reasons that the visibility graph approach is selected as a general method for approaching the basic movers' problem are as follows:

- It is relatively easy to implement.
- It is a fast and reliable method when applied in low-dimensional configuration spaces (i.e. \mathbb{R}^2). Note the configuration space of the AGV in the basic movers' problem is \mathbb{R}^2 .
- When the AGV's configuration space is \mathbb{R}^2 , it establishes optimal paths.
- The visibility graph approach can be extended with no further computational effort to incorporate non-holonomic kinematic constraints of a car-like AGV, producing near-optimal paths, (Jiang *et al*, 1996) and (Jiang *et al*, 1999).
- The visibility graph approach can be relatively easily extended to solve the dynamic motion planning problem establishing optimal motions, see chapter five for details.

Note that current implementations of the visibility graph approach, construct the entire visibility graph or a part of it by only considering the tangential to the obstacles' configuration space edges, thus reducing the size of the visibility graph and therefore

making the search process quicker. The proposed algorithm is called V*MECHA and reduces further the number of vertices considered for the construction of the visibility graph, thus making the process even faster.

The V*MECHA algorithm is based on the V*GRAPH approach proposed by Alexopoulos and Griffin (1992) for solving the basic movers' problem, details of this approach are discussed in section 4.3.2. Note that Conn *et al* (1997), constructed a counterexample showing that the V*GRAPH algorithm is incorrect and therefore is not complete, see section 4.4 for details. In this thesis the V*GRAPH algorithm is extensively studied, an additional deficiency of the algorithm other than that reported by Conn *et al* (1997) is identified and recommendations for the algorithm completeness are made. Additional improvements on the algorithm are proposed, resulting in a new proposed algorithm called V*MECHA, for the path planning of an AGV. Note that proof of the proposed algorithm's completeness and optimality are provided.

The V*MECHA algorithm, in the worse case (this is when the AGV's environment contains only convex obstacles) is not as efficient as other algorithms, such as (Rohnert, 1986) and (Liu and Arimoto, 1992), from an algorithmic theory point of view (but not always from practical point of view). However, in the average case (this is when there are non-convex obstacles in the AGV's environment), which in general appears more frequently in real world applications, the algorithm performs better than the

aforementioned approaches, due the fact that the produced visibility graph is very small and thus the search for a path is quicker³. More details are discussed in chapter four.

Another algorithm is proposed, which is an extension of the V*MECHA algorithm for solving the dynamic motion planning problem for an AGV. This algorithm is called D*MECHA and establishes time-minimal motion for an AGV between two query points in dynamic environments. The algorithm constructs a visibility graph in the space-time configuration space of the AGV and thus reduces the dynamic motion planning problem to that of the static path planning problem. This algorithm is indeed very efficient from both theoretical and practical point of view. The D*MECHA algorithm is extended to handle several types of obstacles' motions.

1.5 Overview of the Thesis

The thesis is organised as follows:

Chapter two presents some definitions, algorithmic notions and mathematical notations that will be used throughout this thesis. This presentation mostly serves as a quick reference to the basic concepts of the aforementioned subject areas, in order to make the concept of the thesis more thoroughly understood.

³ In fact the best result for this problem today is $O(n \log n)$, where n is the total number of the obstacles vertices proposed by Hershberger and Suri (1999). However, even though this is a very good result from a theoretic point of view, the algorithm is not implementable because it is very complex.

In chapter three, a literature survey on robot motion planning approaches that have been proposed to date is presented and a critical review of these is conducted.

Chapter four investigates the V*GRAPH algorithm proposed by Alexopoulos and Griffin (1992) and identifies the problems and deficiencies of the algorithm. Recommendations are made to correct and complete the algorithm and further improvements are considered resulting in the proposition of the V*MECHA algorithm for effectively solving the path planning problem for an AGV.

In chapter five, the motion planning problem for an AGV in dynamic environments is considered. More specifically the problem of planning motions for an AGV operating in a two-dimensional environment populated by linearly moving obstacles is considered and an algorithm for solving it is proposed. This algorithm is an extension of the V*MECHA algorithm presented in chapter four and is called D*MECHA.

In chapter six the applicability of the D*MECHA algorithm in more complicated dynamic environments is discussed. Possible extensions are proposed to enable the D*MECHA algorithm to solve more complicated instances of the motion planning problem for an AGV in dynamic environments. In particular the motion planning problem for an AGV in environments, which contain obstacles that change their size over time and obstacles that have piecewise linear motions are considered.

Chapter seven reviews the thesis, draws some conclusions from the work presented in the thesis, summarises the contributions of the work and makes recommendations for further work.

In Appendix A, an algorithm for computing the obstacles' configuration space, using Minkowski sums is presented. In Appendix B, the A* graph-searching algorithm is presented. Finally, Appendix C contains copies of the publications that have been produced during the course of the work described in the thesis.

1.6 Discussion

In this thesis the challenging problem of motion planning for autonomously guided vehicles is studied. In particular the path planning problem for an AGV in static known environments and the motion planning problem of an AGV in dynamic known environments, are considered. Two approaches are proposed for effectively solving these problems. These approaches in general produce optimum paths and motions and they are suitable for motion planning of an AGV in known and well structured environments, such as, industrial environments, chemical and nuclear plants or even for the navigation of water vessels.

The proposed approaches are experimentally tested through examples, they are critically compared with existing approaches and their advantages and disadvantages are identified and reported.

Here is a brief summary of the contributions of the research described this thesis:

Contributions

1. The V*GRAPH approach proposed by Alexopoulos and Griffin (1992), for solving the basic movers' problem is extensively studied. An algorithmic deficiency other than that presented by Conn *et al* (1997), is identified and reported.
2. Corrections for the completion of the V*GRAPH algorithm are proposed enabling it to solve the path planning problem effectively. Further methodic and algorithmic improvements are made on the V*GRAPH algorithm resulting in the proposition of a new reduced visibility graph approach called V*MECHA algorithm, for solving the basic movers' problem.
3. Proposition of the SUPER_EXTREMES routine, which identifies the super-extremes of the extreme vertices of the visible sequence(s) for each obstacle in order to reduce the size of the visibility graph constructed by the V*MECHA algorithm.
4. Proposition of an algorithm called D*MECHA for solving the motion planning problem for an AGV in dynamic environments.
5. Investigation and proposition of possible extensions to the D*MECHA algorithm to be applicable in more complex dynamic environments are made. Specifically extensions to the D*MECHA algorithm, for solving the motion planning problem for an AGV in dynamic environments populated by obstacles that change their size over time are proposed. Also, extensions to the D*MECHA algorithm, for

solving the motion planning problem for an AGV in dynamic environments populated by piecewise linearly moving obstacles are proposed.

6. Discussion of formal proofs of the proposed algorithms' correction and optimality. Critical comparisons with existing similar algorithms for solving the motion planning problem are also conducted.

1.7 References

- ALEXOPOULOS, C. and GRIFFIN, P. 1992. Path Planning for a Mobile Robot. *IEEE Transactions on Systems, Man and Cybernetics*, **22** (2), pp. 318 – 323.
- CANNY, J. F. 1988. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA.
- CONN, R. A., ELENES, J. and KAM, M. 1997. A Counterexample to the Alexopoulos - Griffin Path Planning Algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, Part B, **27** (4), 721 – 723.
- HERSHBERGER, J. and SURI, S. (1999). An Optimal Algorithm for Euclidean Shortest Paths in the Plane. *SIAM Journal on Computing*, **28** (6), pp. 2215 – 2256.
- HWANG, Y. K. and AHUJA, N. 1992. Gross Motion Planning - A Survey. *ACM Computing Surveys*, **24** (3), pp. 219 - 291.

JIANG, K, SENEVIRATNE, L. D. and EARLES, S. W. E. 1996. Three – Dimensional Shortest Path Planning in the Presence of Polyhedral Obstacles. *Proceedings ImechE, Part C: Journal of Mechanical Engineering Science*, **210** (4), pp. 373 – 381.

JIANG, K, SENEVIRATNE, L. D. and EARLES, S. W. E. 1999. A Shortest Path Based Path Planning Algorithm for Nonholonomic Mobile Robots. *Journal of Intelligent and Robotic Systems*, **24**, pp. 347 - 366.

LATOMBE, J. C. 1991. *Robot Motion Planning*. Massachusetts: Kluwer Academic Publishers.

LIU, Y. and ARIMOTO, S. 1992. Path Planning Using a Tangent Graph for Mobile Robots Among Polygonal and Curved Obstacles. *The International Journal of Robotics Research*, **11** (14), pp. 376 - 382.

LOZANO-PÉREZ, T. 1983. Robot Programming. *Proceeding of the IEEE*, **71** (7), pp. 821 - 841.

LOZANO-PÉREZ, T. and WESLEY, M. A. 1979. An Algorithm for Planning Collision-Free Paths Among polyhedral Obstacles. *Communications of the ACM*, **22** (10), pp. 560 - 570.

ROHNERT, H. 1986. Shortest Paths in the Plane with Convex Polygonal Obstacles. *Information Processing Letters*, **23**, pp. 71 - 76.

SCHWARTZ, J. T. and SHARIR, M. 1983. On the "Piano Movers'" Problem. II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds. *Advances in Applied Mathematics*, **4**, pp. 298 – 351.

SCHWARTZ, J. T. and SHARIR, M. 1988. A Survey of Motion Planning and Related Geometric Algorithms. *Artificial Intelligence*, **37**, pp. 157 - 169.

WAGER, M. L. 2000. Making Roadmaps Using Voronoi Diagrams. [WWW]. <http://www.cs.uwa.edu.au/~michaelw/hons/roadmap.html> (16 November 2000).

YAP, C. K. 1987. Algorithmic Motion Planning. In: SCHWARTZ, J. T. and YAP, C. K. *Advances in Robotics: Algorithmic and Geometric Aspects of Robotics*. Volume 1. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers, pp. 95 - 143.

2

Preliminaries

Now, these are the foundations

II CHRONICLES 3: 3

2.1 Introduction

The intention of this chapter is to present some definitions, algorithmic notions and mathematical notations that will be used throughout this thesis. This presentation is by no means an exhaustive review of algorithmic research, computational geometry, topology or of various aspects of robot motion planning, but it mostly serves as a quick reference to the basic concepts of the aforementioned subject areas, in order to make the concept of the thesis more thoroughly understood. Some notions and definitions have been deliberately omitted from this chapter and are discussed in detail in the following

chapters. In addition, a list of notational conventions used throughout this thesis is provided at the beginning of the thesis.

2.2 Algorithmic Research

The concept of algorithms is a very important area of computer science and is used widely in this thesis. The word algorithm itself is very interesting and over the years many attempts have been made by linguists to find the derivation of the word. For a brief historic review see (Knuth, 1997). According to the definition found in the Webster's dictionary (1995), the name algorithm is given after the Persian mathematician Mohammed al-Khwârizmî (825 AD) and its definition is as follows:

al-go·rithm / 'al-g&-"ri-[th]&m /: a procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation; **broadly**: a step-by-step procedure for solving a problem or accomplishing some end especially by a computer.

A more formal definition of an algorithm is: a well-defined computational procedure, which is composed of a finite number of unambiguous, logical and mathematical steps, which take as an input a (set of) value(s) and produce as an output a (set of) value(s), for solving a given problem (set of problems). In most theoretic computer science textbooks, algorithms are explicitly presented in a pseudo-programming language (or pseudo-code). It is supposed that with a small programming effort this pseudo-code can

be easily transformed into code of a programming language, which is understandable by a compiler. In this thesis, all algorithms will be presented in this manner (pseudo-code).

In general, algorithmic research or algorithmic analysis is not only concerned with establishing an algorithm to solve a given problem but also with solving this problem efficiently in terms of computational effort. This effort can be characterised by means of computational time (also called running time or computational complexity) and computational storage (also called space complexity). For a comprehensive introduction on algorithmic analysis, see (Cormen *et al*, 1990).

Given an algorithm for solving a problem, its computational efficiency is measured in terms of the size of the input of the algorithm. This is because the running time of the algorithm grows with the size of the input. For example, consider the problem of searching an array of integers to find the position in the array of a particular integer I (the integer I appears only once in the array). Suppose that the array has n elements. By using a sequential search to find the position in the array with value I , the algorithm starts from the first element and searches through the array for I until it finds it. The number of operations the algorithm should perform depends on (apart from the actual values to be searched) the number of values forming the input, in this case the size of the array. However, intuitively there can be different running times of the algorithm for different inputs. If the element with value I appear in the first position in the array, the algorithm only examines one element. In this case, the running time is very low and it is said that this is the *best case* for this algorithm. If I is in the last position in the array, the algorithm examines n elements in order to find it and it is said that this is the

worst case for this algorithm. If the sequential search is applied to many arrays of size n , it is expected on average for the search to go half way through the arrays (providing that any position in the arrays has equal probability to host I), therefore on average the algorithm will examine $n/2$ elements. It is then said that this is the *average case* for this algorithm.

In general the best case for an algorithm appears very infrequently and does not fairly characterise the computational behaviour of an algorithm. However, there are some cases where the best case performance of an algorithm is of interest. Analysing and considering the worst case performance of an algorithm is of great interest because it gives an indication that an algorithm could perform at least that well. This is very important in critical applications and in real-time applications, where it is important to be certain about the algorithm's performance for any given input. Often an average case analysis is a more realistic measurement of an algorithm's performance. For instance, when an algorithm is running repeatedly on many sets of inputs. Unfortunately, average case analysis of an algorithm is not always possible because it requires knowledge about the data distribution of the input. In this thesis all algorithms presented or developed will be accompanied by a computational complexity analysis for their worst case performance, which appears to be the most representative and least optimistic measurement of an algorithm's performance.

2.2.1 Asymptotic Analysis

As mentioned in section 2.2, the time efficiency of an algorithm depends on the size of the input of the algorithm. Consider the problem of sorting the elements of an array of

integers, of size n in ascending order. The running time of the algorithm is defined as the maximum number of *primitive operations* required by the algorithm to process the input, which is in this case of size n . Note that with the term primitive operation it is meant an operation, whose execution time is constant and does not depend on its operands. However, it is not very easy especially for complicated algorithms to make an accurate estimation of the number of primitive operations required by the process. Consider that for the sorting problem mentioned above, the *bubble sort* algorithm is used to solve it. After a careful analysis of the algorithm in Knuth (1998), it is derived that the running time of the algorithm in the worst case is defined by the function $f(n) = 7.5n^2 + 0.5n + 1$ over the input (the n elements of the array). Note the worst case is when the initial array appears in descending order. This means that if there are $n = 5$ elements to be sorted, the algorithm can sort them using at most 191 primitive operations. If the number of elements to be sorted increases, it can be noticed that the number of primitive operations used by the algorithm will be different. For very large inputs, the algorithm's running time grows by a factor approximately n^2 , even though the exact number of primitive operations is different. This happens because for large values of n , the low-order terms of the formula and the leading term's constant coefficient are relatively insignificant. For this reason, a simplified analysis is used to estimate the number of operations used by the algorithm for a given input. This is called asymptotic analysis and refers to the analysis of an algorithm, as the input size gets sufficiently large. To be more precise asymptotic analysis is concerned mostly with the growth rate of the number of primitive operations used by the algorithm as the input size gets sufficiently large.

Definition 2.1 (asymptotic upper bound)

If $T(n) \geq 0$ represents the true running time of an algorithm, it is said that $T(n)$ is in $O(f(n))$, if and only if there exist two positive constants c and n_0 such that $T(n) \leq c f(n)$, for all $n > n_0$.

For the worst case of the bubble sort, even though the exact running time is $T(n) = 7.5n^2 + 0.5n + 1$, the term n^2 will dominate the growth as n increases. Therefore it could be said that the algorithm is in (or runs in) $O(n^2)$ time. Note that the upper bound is not the same as the worst case for a given input of size n . It is rather the upper bound for the growth rate of the primitive operations used by an algorithm as the input changes. Thus, it makes sense to define the upper bound of the best case (or the average case or the worst case). Since the bubble sort is in $O(n^2)$ in the worst case, it is true to say that bubble sort is also in $O(n^5)$. For this reason, it is of interest to define, the lowest possible upper bound (tight upper bound). Some textbooks provide a definition of the asymptotic upper bound using equality with the big-Oh notation. It seems more precise to say that $T(n)$ is in $O(f(n))$ rather than $T(n) = O(f(n))$, because there is not a strict equality between $T(n)$ and $O(f(n))$. For instance, $O(n)$ is in $O(n^5)$, but $O(n^5)$ is not in $O(n)$.

Just as an algorithm has an upper bound for a class of input, it also has a lower bound. This bound again bounds the growth rate of the algorithm from below, and is defined as the minimum number of the *primitive operations* required by the algorithm to process a class of input.

Definition 2.2 (asymptotic lower bound)

If $T(n) \geq 0$ represents the true running time of an algorithm, it is said that $T(n)$ is in $\Omega(f(n))$, if and only if there exist two positive constants c and n_0 such that $T(n) \geq c f(n)$, for all $n > n_0$.

It is important to specify for which particular class of input the asymptotic analysis is carried out. For instance, for the *Insertion sort* (Knuth, 1998) the asymptotic tight lower bound is $\Omega(n)$ in the best case for the algorithm and $\Omega(n^2)$ in the worst case for the algorithm. Sometimes the time efficiency of an algorithm is lower bounded by the complexity of the input itself. For example, it is not possible to sort the n elements of an array if all of them are not examined at least once. The input itself can provide some indication for the asymptotic lower bound of an algorithm and it can be said that a sorting algorithm is at least in $\Omega(n)$.

When an algorithm for solving a particular problem attains its lower bound it is said to be optimal, since there is no other algorithm that solves this problem by performing better asymptotically. If the lower bound and the upper bound of an algorithm are, $\Omega(f(n))$ and $O(f(n))$ respectively, it is then said that the algorithm has a tight bound of $\Theta(f(n))$.

Definition 2.3 (asymptotic tight bound)

If $T(n) \geq 0$ represents the true running time of an algorithm, it is said that $T(n) = \Theta(f(n))$, if and only if, $T(n)$ is in $O(f(n))$ and $T(n)$ is in $\Omega(f(n))$.

Often there is a gap between the lowest upper bound of an algorithm and its highest lower bound (provided by the worst case construction). This means that there is either worse worst case construction to be set or there is still room for improvement on the upper bound of the algorithm.

2.2.2 Time and Space

As mentioned in section 2.2, another computing resource besides time that is of concern is the space. The amount of disk space or memory space that is available in a system is a very important consideration in the design of an algorithm. The computational complexity of an algorithm is a measurement of the time that the algorithm itself requires to solve a problem for a given set of data inputs. Space complexity is rather a measurement, which is determined for the data itself. For example, the space requirement for the array of integers mentioned in section 2.2.1 in the example of sequential search is kn bytes providing that each integer requires k bytes for its storage in the array. Thus, this data structure is $\Theta(n)$. There are cases where the running time of an algorithm for solving a particular problem can be reduced by sacrificing the space requirements. This is known as the *space/time trade-off* principle.

2.2.3 Hardness of the Problems

In the computer theoretician community a problem is characterised as "hard" when the best existing algorithm that solves the problem has a very expensive running time. Such problems could be those with exponential running time. A typical example of such a problem is the TOWERS OF HANOI (sometimes referred to as *the tower of Brahma* or *the end of the world puzzle*) described in Harel (1993). In the problem there are three

pegs and a tower of disks on the first peg, with the smallest disk on the top and the biggest on the bottom, Figure 2.1 depicts this problem. The objective of the problem is to move all the disks from the first peg to the third peg with the help of the second peg, by moving only one disk at a time and by not resting a bigger disk on top of a smaller one.

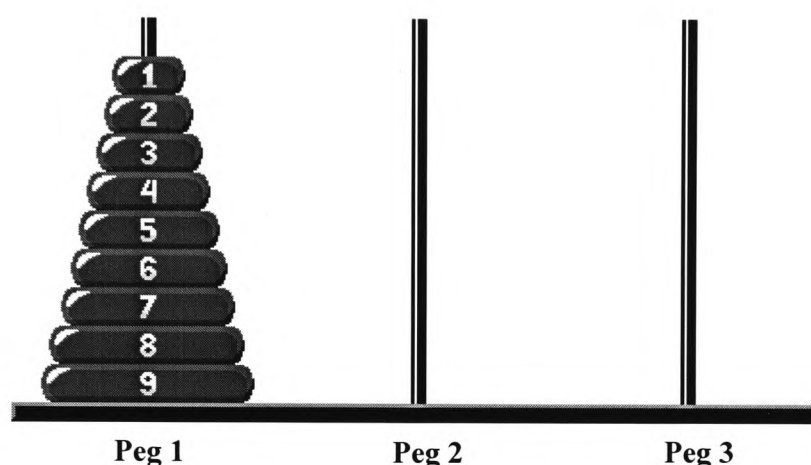


Figure 2.1 Illustration of the TOWERS OF HANOI problem.

This problem is very well understood but the best known algorithm for solving it is in $\Theta(2^n)$. Try to run the algorithm for solving this problem for a "reasonably" large number of disks, e.g.¹ 64, the running time is catastrophically long. If it takes one second to move one disk, it will take 585 billion years to move the 64 disks.

If there is a polynomial time algorithm to solve a problem, that is an algorithm whose running time is in $O(p(n))$, for some polynomial p , with n input length, then it is said that the problem is in P . The distinction between the polynomial and exponential time

¹ The original Tibetan Version of the puzzle involves 64 disks. Actually the Tibetans knew that the solution of the puzzle is extremely time consuming and they believed that the world would end when all the 64 disks were correctly piled up.

algorithms is very important when considering the solution of a problem for large problem instances. Edmonds (1965), in a philosophical digression on the meaning of "efficient algorithm", characterised the polynomial algorithms for solving a problem, "good algorithms" and he pointed out a problem (Graph Isomorphism), which cannot be solved by such "good algorithms". A problem that can be solved by an algorithm of polynomial time, it is referred to as tractable. If a problem is so hard that there is no polynomial time algorithm to solve it, it is then referred to as *intractable* (Garey and Johnson, 1999). Note that not only problems, which require exponential time to be solved, are intractable but also problems for which the solution itself is so extensive that it cannot be bounded by a polynomial function of the input. Allan Turing, (Turing, 1936) presented the first results on *undecidability*. He showed that problems exist, which no algorithm can solve, hence they are called undecidable. Such a problem is the HALTING PROBLEM, the solution to this problem requires an algorithm for deciding whether a given program will ultimately halt for a given input. It is plausible to assume that since problems exist, which no algorithm can solve, these problems are intractable.

A successful approach to deal with the intractable *decidable* problems came from the *non-deterministic Turing machine*. The non-deterministic Turing machine can be imagined as a computer, which has the ability to "guess" the correct solution for a problem from among all its possible solutions. An alternative way to regard a non-deterministic machine is as a super parallel computer, every terminal of which runs the same program (simultaneously) but each gets a different choice (guess)².

² Note that this is not real parallel computing, which allows the computers to run different programs and interact with each other.

If for a given problem, a solution can be "guessed" and this solution can then be checked whether it is correct or not in polynomial time, then this problem can be solved in polynomial time in a non-deterministic machine, even if the number of possible solutions is exponential. It is then said that this problem is in NP (Non-deterministic Polynomial) and algorithms that work in this manner are called *non-deterministic*. This means that a problem in NP can be solved in polynomial time using an infinite number of computers to corroborate in parallel all possible solutions to the problem. Note that not all the problems, which require exponential time in an ordinary computer, are in NP. For instance, the TOWERS OF HANOI problem is not in NP because it is not possible for a non-deterministic machine to "guess" and print out the correct answer in polynomial time. The ability of a non-deterministic machine is limited to "guessing" if a given choice among the set of all solutions is correct. Therefore, only those problems, to which a guessed solution can be checked for its correctness in polynomial time, can be solved in polynomial time by a non-deterministic machine.

If a problem is in NP and all other NP problems can be reduced to it in polynomial time, it is then said that this problem is *NP-complete*. There are many problems known to be NP-complete. For a comprehensive list see in Garey and Johnson (1999). What is really interesting about the NP-complete class is that, if someone ever finds a polynomial time algorithm in an ordinary computer to solve any of the problems in this class, then all the NP problems could be solved in polynomial time using an ordinary computer, by a series of reductions. One of the most important questions with no answer in computer science is whether $P = NP$. Figure 2.2, illustrates a view of the

interrelation of the problems, which their solution requires at most exponential time algorithms.

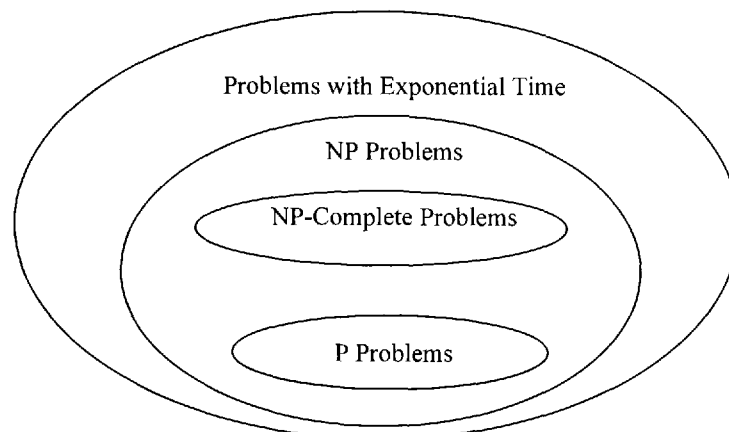


Figure 2.2 Relation between the problems, whose solution requires exponential time algorithms.

If there is a NP problem, say X , and it can be reduced in polynomial time to a problem Y , then problem Y is said to be *NP-hard*. Alternatively, a problem is NP-hard, if it is at least as hard as any NP problem. The NP-hardness of a problem is determined by a series of transformations of the problem to one of the known NP-hard problems. If there is an NP problem, say X , such that Y is in polynomial time reducible to it, then Y is NP-easy. For a more detailed discussion on NP-completeness, see (Papadimitriou and Steiglitz, 1982), (Van Leeuwen, 1990) and (Garey and Johnson, 1999).

Another class of problem's hardness is the *PSPACE*. A problem is said to be in PSPACE if it can be solved by an algorithm of polynomial space complexity on a non-deterministic machine (hence on a deterministic machine). Similar definitions to NP-completeness and NP-hardness, apply to PSPACE-completeness and PSPACE-hardness

respectively. If there is a PSPACE problem say X , and it is polynomially reducible to a problem Y , then problem Y is said to be *PSPACE-hard*. If a problem is in PSPACE and is PSPACE-hard, it is then said that this problem is *PSPACE-complete*. The ability to subsume a problem in one of the above classes provides hard evidence for the lower bound of the computational complexity of the problem.

2.3 Computational Geometry and Topology

Computational geometry is the branch of computer science, which studies algorithms for the solution of geometric problems, (Mulmuley, 1994), (de Berg *et al*, 1997), (O'Rourke, 1998) and (Boissonnat and Yvinec, 1998). The input to a typical computational geometric problem is the description of a (or a set of) geometric object(s), such as points, line-segments, polytopes and so forth. The output is often the construction of a geometric structure such as the convex hull of a set of points, or it could be the response to a query about the input, such as whether the boundaries of two polygons intersect.

Many techniques for solving the robot motion planning problem use a geometric representation of the robot's environment. Such a geometric representation is called a geometric map and is made up of discrete geometric primitives such as points, lines, polygons, polyhedra and so forth. Note that the robot's space cannot only be geometrically represented. Another way to represent the space is by discretely sampling the space itself using a grid or another spatial decomposition tool and expresses the degree of occupancy of the sample points (cells of the grid). The robot's space can also

be topologically represented. The topological representations rely explicitly on the connectivity between regions or objects and in contrast with the geometric representation involve absence of metric data.

In this section, some definitions and notational conventions from computational geometry and topology are presented.

The set of the real numbers is denoted by \mathbb{R} . This set can be represented by the real number line. For points p and q respectively on the real line, with $p \leq q$, the *closed* interval, which includes p and q is denoted by $[p, q]$ and the *open* interval, which does not include them is denoted by (p, q) , (in some textbooks this can be found as, $]p, q[$). A d -dimensional space of reals is defined by \mathbb{R}^d . Imposing the Euclidean metric on the real space \mathbb{R}^d creates a Euclidean space E^d . This metric is the standard distance function. For two points p and q in the Euclidean plane E^2 with co-ordinates (x_p, y_p) and (x_q, y_q) respectively the standard Euclidean distance between p and q is obtained by equation (2.1).

$$d(p, q) = \|p - q\| = \sqrt{(x_q - x_p)^2 + (y_q - y_p)^2} \quad (2.1)$$

In the same manner, a subset S of \mathbb{R}^d , is *closed* if it includes its *boundary* ∂S and *open* if it does not. The *interior* of a set S , $\text{int}(S)$, is the set S without its boundary ∂S . The *complement* \bar{S} , of a set S , consists of all points in the space which are not in S . The *closure*, $\text{cl}(S)$, of a set S contains both its interior and its boundary and is $\text{cl}(S) = \text{int}(S) \cup \partial S$. A set S is called *compact* if it is both closed and bounded. A set is

said to be *connected* if any two points $p, q \in S$, can be connected with a curve, and this curve is contained in S , otherwise it is called *disconnected*.

2.3.1 Higher-Dimensional Spaces

Points on the plane and in the three-dimensional space can be specified by means of Cartesian co-ordinates. Many notions and properties of two- and three - dimensional geometry can be generalised to spaces of n -dimensions with $n > 3$. For instance if the points p and q have (p_1, p_2, \dots, p_n) and (q_1, q_2, \dots, q_n) co-ordinates respectively in the n -dimensional space, the distance between p and q is obtained by equation (2.2).

$$d(p, q) = \|p - q\| = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2} \quad (2.2)$$

Hyperplane is a generalisation of a line in the n -dimensional space. Therefore, the set of points whose co-ordinates satisfy the linear equation (2.3) is called a hyperplane and divides the n -dimensional space into two half-spaces, as the plane divides the three-dimensional space in two half spaces.

$$a_1x_1 + a_2x_2 + \dots + a_nx_n = b \quad (2.3)$$

A *hypersurface* is a $(n-1)$ -dimensional set that generalises a curve. If a hypersurface can be described by a polynomial of bounded degree, it is said that the hypersurface has a bounded algebraic degree.

2.4 Configuration Space and Degrees of Freedom

This section describes some very important issues for solving the robot motion planning problem.

The physical space in which a robot operates is most of the time equal to the Euclidean space of dimensions two or three (E^2 or E^3) and it is referred to as the *workspace*. A robot's *degrees of freedom (dof)* are equal to the number of independent parameters that are adequate to describe the specification of any placement of the robot in the workspace. Using an artificial space the *configuration space* (or *joint space*) of dimensions equal to the number of the dof of the robot, every configuration of the robot can be represented uniquely within this space as a point. Figure 2.3 illustrates two different types of robots, a polygonal rigid robot and an articulated arm, each of them having a different number of degrees of freedom.

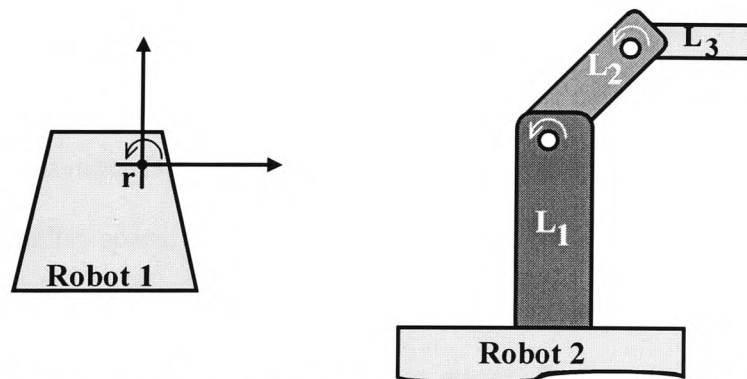


Figure 2.3 Illustration of different types of robots. On the left there is a polygonal robot with three degrees of freedom and on the right there is a robotic manipulator with two degrees of freedom.

Robot 1 is a *simple*³ polygonal rigid body, which can translate freely in the x-y plane, resulting in two dof and rotate about its reference point r , adding another degree of freedom. The total number of the dof of Robot 1, is three and its configuration space is $\mathbb{R}^2 \times [0, 2\pi)$. Robot 2, is an articulated arm, which moves in a two-dimensional workspace (hence, sometimes is called planar arm). It has three links L_1 , L_2 and L_3 . The first link is mounted on the ground and the other two links are connected to each other with two revolute joints. A placement of Robot 2, can be uniquely defined by the angle of the two joints. Therefore, Robot 2 has two degrees of freedom. Suppose that the links do not collide with each other at any placement of the arm, then any placement of the arm can be defined by any pair of joints' angles (note that any pair of angles belong to the $[0, 2\pi)^2$). The configuration space of Robot 2 is $SO(2)$. $SO(n)$ is the Special Orthogonal Group of the $n \times n$ matrices in \mathbb{R}^{n^2} , with orthonormal columns and determinant equal to +1.

The idea of reducing the dimension of the robot into a point in an artificial space was first introduced by Udupa (1977). Later, Lozano-Pérez and Wesley (1979), used this idea and represented a robot as a point (point-robot) in the configuration space and they mapped the environment's obstacles into that space. When the obstacles were mapped in the configuration space, they then planned the motion of the point-robot into this space. This algorithm finds collision-free motions of polygonal/polyhedral robots among polygonal/polyhedral obstacles. Using the configuration space approach they reduced the problem of finding a path between two points for a dimensioned robot among dimensioned obstacles to that of finding a path between two points for a point-

³ A polygon is said to be simple when its edges intersect only at their end points.

robot among dimensioned obstacles. Lozano-Pérez and Wesley's algorithm is considered as the first exact robot motion planning algorithm. Since then the idea of planning robot motions in a configuration space has been used widely.

2.4.1 Configuration Space of a Robot

Consider the workspace $W = \mathbb{R}^n$, with $n = 2$ or 3 , populated by dimensioned physical obstacles P_i , where $i \in \mathbb{N}$ and the rigid robot R in a specified position in W with a certain orientation. Further consider that F_W a Cartesian co-ordinate (global) frame attached to W with origin O_W and F_R a Cartesian co-ordinate (local) frame attached to R with origin O_R respectively (F_W is fixed while F_R is moving). A *configuration* q of R is the specification of the position and orientation of F_R with respect to F_W . All the configurations of R constitute the *Configuration Space* C (or *Cspace*) of R . When R is not a rigid body, for instance when it is an articulated manipulator, then for every body R_i , $i \in \mathbb{N}$ of R , a Cartesian co-ordinate frame F_{R_i} is attached to it with origin O_{R_i} . A configuration q_i of R is the specification of the position and orientation of every F_{R_i} with respect to F_W . All the configurations q_i of R constitute the Configuration Space C of R . The configuration space of a robot can be a *multiple connected* space, this means that there could be more than one path connecting two configurations in it. In robot motion planning, a function often is defined in order to refine the paths in the configuration space and choose one of them, depending on what sort of path (i.e. shortest path, time-minimal path or so on) is required. In Latombe (1991), the notion of configuration space is discussed in greater detail.

2.4.2 Mapping the Obstacles in the Configuration Space

If $R(q)$ denotes the subset of W occupied by R at configuration q then $P_i, i \in [1, \dots, p]$ maps from W in C to an area CP_i , which is called *Obstacle's Configuration Space* or *C-Obstacle*. More formally:

$$CP_i = \{q \in C \mid R(q) \cap P_i \neq \emptyset\} \quad (2.4)$$

This area represents the configurations, which are illegal for the robot to attempt, because it will collide with the obstacle P_i . The union of all obstacles' configuration spaces CP_i is called *C-Obstacle region* and is formally defined as:

$$\bigcup_{i=1}^p CP_i \quad (2.5)$$

This region represents all the inaccessible configurations for the robot to attempt, because in these configurations the robot will collide with some obstacles. In section 2.4.4, a method for computing the configuration space of the obstacles is presented.

2.4.3 Path in the Configuration Space

A path for a robot R between the configurations q_{start} (robot's start configuration) and q_{goal} (robot's goal configuration) in C can be defined as a continuous map:

$$\tau : [0, 1] \rightarrow C, \text{ where } \tau(0) = q_{\text{start}}, \tau(1) = q_{\text{goal}} \quad (2.6)$$

It is important to note that the topology of C is not always the same as that of W . In the Euclidean workspace W , if two paths connect the same two points, one can be continuously deformed to the other one. However, this is not the case in C . If C is the configuration space and $\bigcup_{i=1}^p CP_i$ is the union of all C-Obstacles, then the collision *free space* C_{free} is defined as:

$$C_{\text{free}} = C \setminus \bigcup_{i=1}^p CP_i = \{q \in C \mid R(q) \cap \bigcup_{i=1}^p CP_i = \emptyset\} \quad (2.7)$$

Any configuration on C_{free} is called *free configuration*, C_{free} space is an open set of all free configurations of R . Any path on C_{free} is called *free path* and it is guaranteed that while a robot is moving along it, it does not intersect or come in contact with any obstacle. A path between two configurations is called *semi-free* path when it is a continuous map:

$$\tau = [0, 1] \rightarrow \text{cl}(C_{\text{free}}), \quad \text{where } \tau(0) = q_{\text{start}}, \tau(1) = q_{\text{goal}} \quad (2.8)$$

While the robot is moving on a semi-free path, it touches the boundaries of the obstacles without intersecting their interior. Motions in contact are also called compliant motions and sometimes result in more robust algorithms, (Avnaim *et al*, 1988).

2.4.4 Computing the Obstacles' Configuration Space

There are several methods for computing the C-Obstacles, seven of them are listed in (Hwang and Ahuja, 1992). These are, point evaluation, Minkowski sums, boundary equation, needle, sweep volume, template and Jacobian-based method. Kavraki (1995),

presented an alternative way for computing the C-Obstacles using Fast Fourier Transform. All these methods can be used for any type of robot and they all work very well in configuration spaces of low dimensions, but as the number of degrees of freedom of the robot increases the configuration space grows exponentially and thus the computation of the C-Obstacles is an extremely difficult task. However note that there are some advantages and disadvantages to each method when used in different environments. For example the Minkowski sums is very straightforward when it is used for the construction of the C-Obstacles when the robot and the obstacles are two-dimensional polytopes (especially for non-rotating robots) and therefore this method is most popular with AGVs. This thesis is mostly concerned with the development of motion planning algorithms for AGVs, therefore in this chapter, the Minkowski sums method will be discussed.

The process of computing the configuration space of the obstacles is called *growing obstacles*. This is because the size of the obstacles is enlarged with respect to the size of the robot and the size of the robot is reduced to a point (point-robot). As long as the point-robot (reference point) is outside of the boundaries of the grown obstacles (C-Obstacles) the robot lies in a collision-free space. Figure 2.4 illustrates the principle for calculating the C-Obstacles when the workspace $W = \mathbb{R}^2$. Suppose that R is a polygonal robot, which translates freely without rotation and the origin of the frame F_R is an arbitrarily chosen reference point on the robot. If P is a polygonal obstacle then CP is the *grown obstacle* or *Obstacle's Configuration Space* (shaded area) and the robot can now be considered as a point.

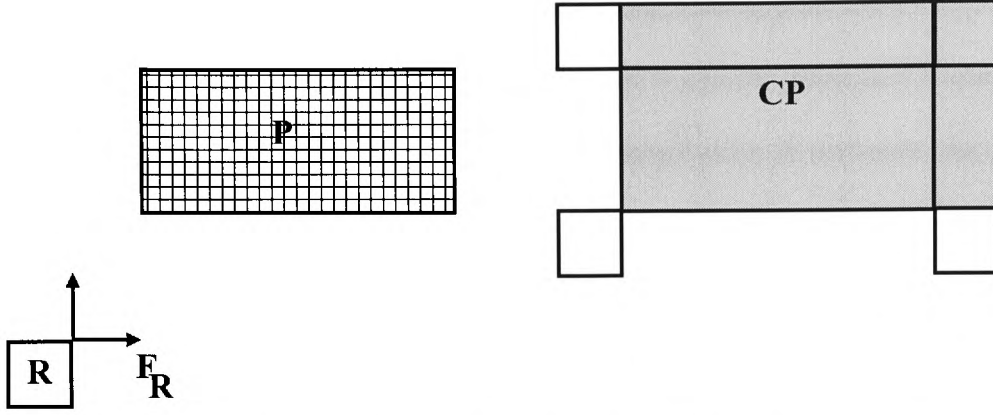


Figure 2.4 The shaded areas constitute the grown obstacle or the obstacle's configuration space CP.

The above process uses the Minkowski sums to calculate the C-Obstacles. The Minkowski sum of two $S_1 \subset \mathbb{R}^2$ and $S_2 \subset \mathbb{R}^2$, is denoted by $S_1 \oplus S_2$ and is defined as follows, (Lozano-Peréz and Wesley, 1983).

For two sets $S_1 \subset \mathbb{R}^2$ and $S_2 \subset \mathbb{R}^2$, the Minkowski sum is,

$$S_1 \oplus S_2 = \{a + b : a \in S_1, b \in S_2\} \quad (2.9)$$

In order to express the C-Obstacles as Minkowski sums, one more notation it is required. If q is a point $q = (q_x, q_y)$, then $-q$ is defined as $-q = (-q_x, -q_y)$ and for a set S , $-S$ is defined as $-S = \{-q : q \in S\}$. As it can be noticed, $-S$, is obtained by reflecting S about its origin.

Visually the C-obstacles can be obtained by arbitrarily choosing a point on the robot as a *reference point* and then keeping a track of the locus of the reference point, while the

robot is moving (by touching) at fixed orientation along each side of every obstacle in the environment. More formally, given that the robot R and the obstacle P are sets in \mathbb{R}^2 , $CP = P \oplus (-R(0, 0))$, where $R(0, 0)$ is the reference point of the robot, the proof can be found in (Lozano-Peréz and Wesley, 1983). If P and R are both convex polygons on the plane the resulting Minkowski sum of their vertices is a convex polygon. If P and R are two-dimensional n -gon and m -gon respectively then the Minkowski sum $P \oplus R$ can be computed in $O(n + m)$ computational time when both P and R are convex, in $O(nm)$ when one of them is convex and the other non-convex and in $O(n^2m^2)$ when both of them are non-convex, (de Berg *et al*, 1997). Laumont, (1987) extended the algorithm for computing the obstacles' configuration space, to be applicable to *generalised polygons*⁴, without loss of the algorithmic complexity. In Appendix A, an algorithm for computing the obstacles' configuration space for a non-rotating convex robot and a convex obstacle is presented. In addition, an extension to this algorithm for non-convex robot/obstacle is discussed along with an analysis of their computational time.

2.5 Discussion

Robot motion planning is a well studied area of robotics, which combines principles from many disciplines such as Algorithmic Research, Computational Geometry, Topology, Mathematics and so on. The aim of this chapter was to summarise some important aspects from these disciplines and to bring up some issues used in the thesis

⁴ Generalised polygons are two-dimensional polytopes, whose boundaries consist of straight line-segments and/or circular arcs.

in order to make the study of the thesis at hand more efficient and its concept more thoroughly understood. This chapter is by no means a complete reference but just a representative source of information of the aforementioned aspects.

2.6 References

- AVNAIM, F., BOISSONNAT, J. D. and FAVERJON, B. 1988. A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles. *Proceedings of the 5th IEEE International Conference on Robotics Automation*. pp. 1656 - 1661.
- BOISSONNAT, J. D. and YVINEC, M. 1998. *Algorithmic Geometry*. English Edition. UK: Cambridge University Press.
- CORMEN, T. H., LEISERSON, C. E. and RIVEST, R. L. 1990. *Introduction to Algorithms*. Cambridge, Massachusetts: The MIT Press.
- DE BERG, M., VAN KREVELD, M., OVERMARS, M. and SCHWARZKOPF, O. 1997. *Computational Geometry, Algorithms and Applications*. Berlin: Springer and Verlag.
- EDMONDS, J. 1965. Paths Trees and Flowers. *Canadian Journal of Mathematics*, **17**, pp. 449 - 467.

GAREY, M. R. and JOHNSON, D. S. 1999. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. New York: Freeman.

HAREL, D. 1993. *Algorithmics, The Spirit of Computing*. 2nd ed. Addison - Wesley.

HWANG, Y. K. and AHUJA, N. 1992. Gross Motion Planning - A Survey. *ACM Computing Survey*, **24** (3), pp. 219 - 291.

KAVRAKI, L. E. 1995. Computation of Configuration-Space Obstacles Using the Fast Fourier Transform. *IEEE Transactions on Robotics and Automation*, **11** (3), pp. 408 – 413.

KNUTH, D. E. 1997. *The Art of Computer Programming, Fundamental Algorithms*. Volume 1. 3rd ed. Reading MA: Addison-Wesley.

KNUTH, D. E. 1998. *The Art of Computer Programming, Sorting and Searching*. Volume 1. 2nd ed. Reading MA: Addison-Wesley.

LATOMBE, J. C. 1991. *Robot Motion Planning*. Kluwer Academic publishers.

LAUMOND, J. P. 1987. Obstacle Growing in a Polygonal World. *Information Processing Letters*, **25** (1), pp. 41 - 50.

- LOZANO-PÉREZ, T. and WESLEY, M. A. 1979. An Algorithm for Planning Collision-Free Paths Among polyhedral Obstacles. *Communications of the ACM*, **22** (10), pp. 560 - 570.
- LOZANO-PÉREZ, T. and WESLEY, M. A. 1983. Spatial Planning: A Configuration Space Approach. *IEEE Transactions on Computers*, **C-32** (2), pp. 108 - 119.
- MULMULEY, K. 1994. Computational Geometry, An Introduction Through Randomized Algorithms. New Jersey, USA: Prentice - Hall.
- O' ROURKE, J. 1998. *Computational Geometry in C*. 2nd Ed. UK: Cambridge University Press.
- PAPADIMITRIOU, C. H. and STEIGLITZ, K. 1982. *Combinatorial Optimisation: Algorithms and Complexity*. Englewood Cliffs: Prentice Hall.
- TURING, A. 1936. On Computable Numbers with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, **2** (42), pp.230 - 265.
- UDUPA, S. 1977. *Collision Detection and avoidance in Computer Controlled Manipulators*. Ph.D. Thesis, Department of Electrical Engineering, California Institute of Technology.

VAN LEEUWEN, J. 1990. *Handbook of Theoretical Computer Science, Algorithms and Complexity*. Volume A. Netherlands: Elsevier Science Publishers.

WEBSTER'S - MERRIAM DICTIONARY OF ENGLISH USAGE. 1995. 2nd edition.
US.

3

Robot Motion Planning - Literature Survey

I have declared the former things from the beginning

ISAIAH 48: 3

3.1 Introduction

As was mentioned in the introduction to the thesis (chapter one), motion planning is considered a central field in the development of robotic systems and it has attracted a great deal of research activity for the past two decades. The motivation behind the research efforts arises from the increasing demand for robotic systems and especially for AGVs from industry. One of the most essential tasks that a mobile robot should undertake in order to be autonomous is to plan its own motions. Therefore, the solution

to the motion planning problem plays an important role in the development of autonomously guided vehicles and robotic systems in general. Various motion planning techniques for solving the basic movers' problem and various extensions of it have been developed over the years, each of them posing its own advantages and disadvantages in different application domains.

In this chapter several important contributions are presented, these contributions are summarised in three main approaches. The *roadmap approaches*, the *cell decomposition approaches* and the *potential field approach*, (Latombe, 1991). Note that these approaches are not mutually exclusive and sometimes a combination is used to solve a particular motion planning problem more effectively.

It is impossible to refer to all the contributions from the last two decades and this is not within the scope of this chapter. Therefore, this chapter does not serve as an exhaustive survey but only gives an indication of the development of motion planning techniques over the past two decades. Further surveys of motion planning can be found in (Akman, 1987), (Yap, 1987), (Schwartz and Sharir, 1988), (Latombe, 1991), (Hwang and Ahuja, 1992a), (Sharir, 1995), (Latombe, 1999) and (Wager, 2000). Note that chapters six and seven are concerned with the dynamic motion planning problem. Brief surveys of approaches used for solving the dynamic motion problem are provided at the beginning of each of these two chapters.

3.2 Roadmap Approaches

The fundamental idea behind the roadmap approaches is to construct a network of one-dimensional curves lying in the robot's collision-free space, C_{free} , or its closure, $\text{cl}(C_{\text{free}})$, in order to capture its connectivity. This network is called a roadmap and once it is constructed then the robot's start configuration, q_{start} , and its goal configuration, q_{goal} , are connected to it, each of them with one-dimensional curve(s), which also lie in C_{free} or $\text{cl}(C_{\text{free}})$. Once the roadmap, R , is constructed and the q_{start} and q_{goal} are connected to it, the path planning problem has been reduced to a graph search problem. Thereby, what is required to solve the path planning problem is to search the one-dimensional network for a path from the robot's start configuration to its goal configuration. The essential matter of the roadmap approaches is the construction of the roadmap itself. There are various types of roadmaps based on different principles. These roadmaps are the *visibility graph*, the *Voronoi diagram*, the *freeway nets*¹, the *silhouettes* and the *probabilistic roadmap*. Each of them will be briefly discussed in the following sections.

3.2.1 Visibility Graph

The visibility graph approach is considered as one of the earliest robot path planning methods. It was first used by Nilsson (1969), to plan the motion of a mobile robot system (the Shakey). A model of the workspace was used and the actions of the robot were stated in a language of first-order predicate calculus. Lozano-Pérez and Wesley (1979), further developed the idea of visibility graph and reported an algorithm for path planning of a polygonal/polyhedral robot in a two/three-dimensional environment.

¹ In some literature this method is subsumed in the cell decomposition approaches, here the taxonomy from (Latombe, 1991) is adopted and the method is subsumed in the roadmap approaches.

The visibility graph approach is mainly used in two-dimensional environments populated by polygonal obstacles. The visibility graph is an undirected graph constructed by considering all the vertices of the C-Obstacles, the robot's start configuration and its goal configuration. Edges of this graph are the line-segments, which connect all the mutually visible vertices (hence visibility graph). Two vertices are mutually visible when they can be connected with a straight line-segment and this segment does not overlap the interior of any C-Obstacle. More formally the undirected graph $VG(V, E)$ is defined as follows:

V is the set of all C-Obstacles vertices as well as the q_{start} and q_{goal} .

E is the set of all edges $e_{ij} = (v_i, v_j)$ such that,

$$\forall i, j \in V \wedge \forall k \in [1, \dots, p] \mid e_{ij} \cap \text{int}(CP_k) = \emptyset.$$

Figure 3.1 illustrates the concept of visibility graph in an environment populated by polygonal obstacles.

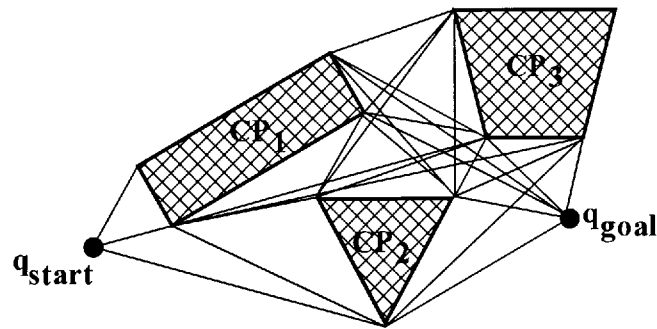


Figure 3.1 Visibility graph with polygonal obstacles.

After the visibility graph has been constructed then a standard shortest path algorithm for weighted graphs can be applied to establish the shortest Euclidean semi-free path.

The algorithm presented in (Lozano-Pérez and Wesley, 1979) is called VGRAPH and it was not accompanied by an upper bound, but it is suspected that is in $O(n^3)$, where n is the total number of the C-Obstacles' vertices, (Canny, 1988). Once the visibility graph was constructed they used heuristics to search it for the shortest path. In particular the A* algorithm due to Hart *et al* (1968) was used, which requires $O(n^2)$ time, where n is the total number of the graph's vertices. Therefore, the bottleneck of their approach is the construction of the visibility graph.

An algorithm for constructing the visibility graph in time $O(n^2 \log n)$, where n is the total number of the obstacles' vertices in the scene, was proposed by Lee (1978). This algorithm can lead to an overall computational complexity of the VGRAPH approach to $O(n^2 \log n)$ where n is the total number of the obstacles' vertices. (Welzl, 1985), (Asano *et al*, 1985) and (Edelsbrunner, 1987) proposed algorithms for constructing the visibility graph in time $O(n^2)$, where n is the total number of the obstacles' vertices. Ghosh and Mount (1987), proposed an output-sensitive algorithm for constructing the visibility graph in $O(k + n \log n)$ time, where k is the number of edges of the visibility graph and n is the total number of the obstacles' vertices.

Establishing the shortest path is fundamental in robot motion planning and is one of the most important issues. The visibility graph method is a very effective method when applied to translational motions in the plane and in general establishes optimal paths

(using Euclidean metric). However, some drawbacks occur when using this method.

These drawbacks are:

- Since the path established by this method lies in the $cl(C_{free})$, the robot in general will touch the environment's obstacles, while it is moving along the path. Therefore, this method does not give rise to safe paths. However, this drawback can appear in many path planning approaches especially to these, which plan the robot's paths in the $cl(C_{free})$ space. Note that any semi-free path can be transformed into a free path, which can be made arbitrarily close to the semi-free path.
- The path obtained using this method is a set of straight line-segments, which give rise to curvature discontinuities. This means that while the robot is moving along this path it might have to slow down completely in order to undertake some turns, which is not time efficient. However, this problem arises in many path planning approaches and one way to tackle it is to smooth out the path once it is found. The smoothing out procedure is at the expense of computational time.
- The optimality of the visibility graph method is not as effective when it is applied in configuration spaces with more than two dimensions. For example, in a three-dimensional configuration space populated by polyhedral obstacles, the shortest path does not in general go through the obstacles' vertices.

Canny (1988) showed that the problem of finding the shortest path in a three-dimensional configuration space, which contains polyhedral obstacles, is NP-hard.

Lozano-Pérez and Wesley (1979) attacked this problem by adding fake vertices in the C-Obstacles' edges so no resulting edge was greater than a predefined length. This method results in a good approximation of the shortest path but it makes the search of the graph more time consuming due to the increased number of vertices. Papadimitriou (1985), proposed a polynomial approximation scheme for solving this problem by breaking the original C-Obstacles' edges in the scene into short segments, he then constructed a visibility graph using these segments as nodes of the graph. The path obtained by the algorithm is at most $(1 + \epsilon)$ times longer than the shortest path. The computational time of the algorithm is polynomial in n and ϵ , where n is the total number of the elements of the polyhedral scene (vertices, edges and faces) and ϵ is the desired accuracy of the approximation algorithm. Jiang *et al* (1996), proposed an algorithm for finding the shortest path between two query points in a three-dimensional environment populated by convex polyhedral in time $O(n^3 v^k)$, where n is the total number of the obstacles' vertices, v is the maximum number of vertices on any one obstacle and k is the number of obstacles. Their approach is based on the concept of the visibility graph. A set of visible boundary edges (VBE) from a given view point are identified using a technique based on projective relationships. The algorithm starts from the robot's initial point as a view point and recursively constructs an initial reduced visibility graph through points on the VBEs at every recursive call, until the goal point is reached. An optimisation technique is used to revise the turning points of each path on the VBEs and the global shortest path is then selected from the three-dimensional reduced visibility graph.

Laumond (1987), extended the visibility graph to be applicable in environments populated by generalized polygons using the properties of the Minkowski operators.

The visibility graph technique was further extended by, Rohnert (1986), Rohnert (1988), Liu and Arimoto (1991), Liu and Arimoto (1992) and Liu and Arimoto (1995) to the *Tangent Graph (T-graph) (or reduced visibility graph)*. The tangent graph is a graph whose vertices are the C-Obstacles' vertices as well as the robot's start and goal points. The edges of this graph are the C-Obstacles' edges as well as the edges connecting all the mutually visible vertices and define common tangents to the C-Obstacles. Figure 3.2 illustrates the tangent graph, where it can be noticed that the number of edges of the tangent graph is considerably reduced from its corresponding visibility graph shown in Figure 3.1. In (Liu and Arimoto, 1992), it was shown that the T-graph contains the shortest semi-free path between two points. Therefore, since the T-graph has fewer edges than the visibility graph, the search for the shortest path is less time consuming.

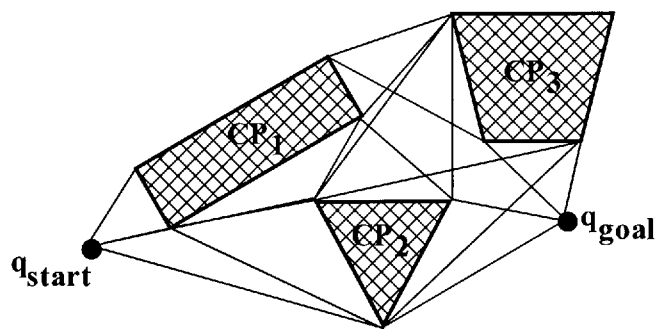


Figure 3.2 Edges of the Tangent Graph are only the cotangents of any pair of the scene's C-Obstacles.

Alexopoulos and Griffin (1992), proposed an algorithm named V*GRAPH for solving the basic movers' problem, in $O(n^2 \log n)$, where n is the total number of C-Obstacles' vertices. The algorithm constructs a reduced visibility graph in a different manner to this of the tangent graph and produces the shortest semi-free path between the AGV's start and goal locations. However, Conn *et al* (1997), constructed a counterexample, showing that the V*GRAPH algorithm is neither complete nor optimal in a sense that finds the shortest semi-free path between two query points as it was claimed by the authors. In chapter four the T-graph approaches will be discussed in more detail and the V*GRAPH algorithm will be studied in order to identify its deficiencies.

3.2.2 Generalised Voronoi Diagram

This approach is also called the *retraction approach*, because the Voronoi diagram is a mapping of the C_{free} ($= \mathbb{R}^2$) space into one-dimensional curves, which also lie in the C_{free} . The mapping has to be continuous and preserve the original space's connectivity. Such a function in topology, which maps an n -dimensional space \mathbb{R}^n , onto an $n-1$ -dimensional space \mathbb{R}^{n-1} is called a retraction function (Preparata and Shamos, 1985), hence the name of the approach. For a detailed survey on Voronoi diagrams, see (Aurenhammer, 1991). A good textbook on the concept and applications of the Voronoi diagram is (Okabe *et al*, 2000). A review of the basic properties of the Voronoi diagram, along with some efficient techniques for its construction can be found in (Leven and Sharir, 1987). Some very brief but very interesting historical remarks about the Voronoi Diagram can be found in (de Berg *et al*, 1997).

The Voronoi diagram (also known as *Dirichlet tessellation* or *Thiessen tessellation*)² of a finite set of points (sites) S in the plane consists of a finite set of points (or Voronoi vertices) and a finite set of edges (or Voronoi edges) that partition the plane into (possibly unbounded) convex regions, which are called Voronoi cells. Each of these cells contains only one site and any point inside this cell is closer to the site enclosed by this cell than to any other site, under a predefined metric (usually L_2). Figure 3.3 illustrates the concept of Voronoi diagram $\mathcal{V}(S)$ of a set of sites S in E^2 .

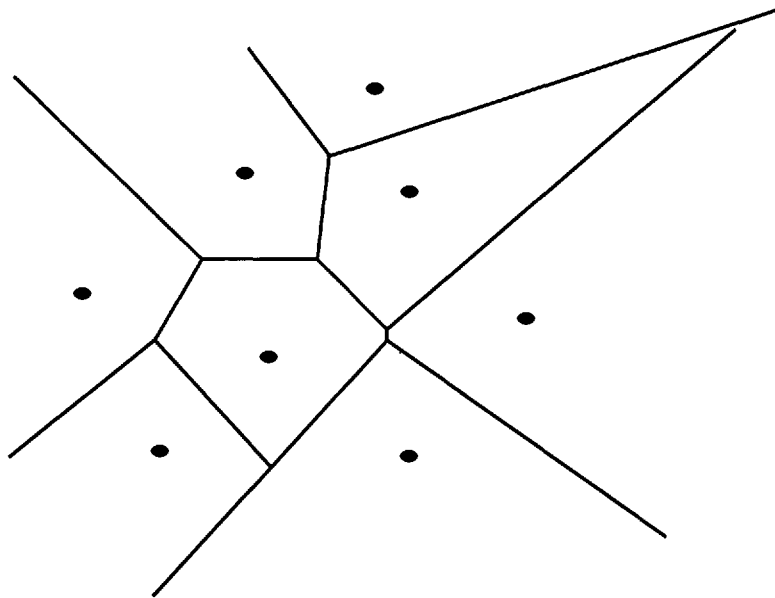


Figure 3.3 Voronoi diagram for eight sites in the plane.

Every edge of the Voronoi diagram is the locus of the points, which are equidistant from at most two sites (bisector) and every vertex of the Voronoi Diagram is equidistant from at least three sites. It is not hard to show using the *Euler's formula* (which states

² The name Voronoi Diagram, Dirichlet tessellation or Thiessen tessellation, has been given to this Geometric structure in honour to the two Mathematicians and one Climatologist respectively for their great contributions regarding this Geometrical construct, (Aurenhammer, 1991).

that for any connected planar embedded graph with v vertices, e edges and f faces the following holds: $v-e+f=2$) that the Voronoi diagram has $O(n)$ edges and $O(n)$ vertices. The lower bound for computing the Voronoi diagram $\Omega(n \log n)$ time, (de Berg *et al*, 1997). Fortune (1986) proposed an algorithm for constructing the Voronoi Diagram in $O(n \log n)$ time and $O(n)$ space, hence his algorithm is optimal. The Voronoi diagram as a geometric data structure was first proposed for solving the robot path planning problem by Rowat (1979).

A *generalised Voronoi diagram (GVD)* of set of points, line-segments, circles and polygons in the plane, was explored by Drysdale (1979). A GVD of a set of points and line-segments in the plane, in general partitions the plane into a complex of non-convex cells, (Lee and Drysdale, 1981). The edges of the GVD are straight line-segments and parabolic-segments.

In robot path planning the GVD is used to construct a roadmap in the robot's C_{free} . When the $C_{\text{free}} = \mathbb{R}^2$, the result $\text{GVD}(C_{\text{free}})$ is a roadmap, which is composed of straight line-segments and parabolic-segments, when sites are the edges and the vertices of the scene's polygons. Figure 3.4 illustrates the GVD of a two-dimensional bounded configuration space populated by polygonal C-Obstacles.

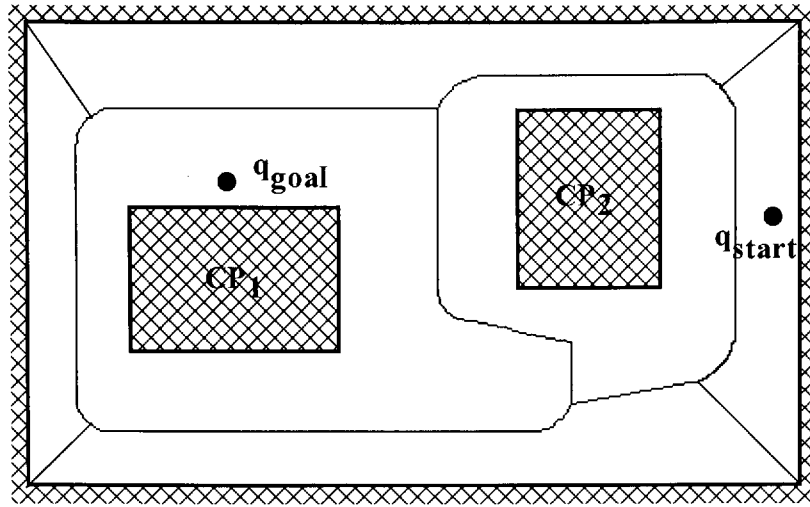


Figure 3.4 The Generalised Voronoi Diagram for a set of convex obstacles in the plane bounded by an external polygonal obstacle.

The straight line-segments in the GVD are sets of points, which are equidistant from two edges or two vertices of the environment's object features and the parabolic-segments are sets of points, which are equidistant from one edge and one vertex of the environment's object features, whose locus and directrix are the vertex and the edge respectively. Note that a GVD of n points and line-segments, has $O(n)$ vertices and $O(n)$ edges (Kirkpatrick, 1979).

As with the visibility graph, the construction of the GVD is a very important issue in the retraction approach for robot path planning and can determine the overall computational time of the approach. Lee and Drysdale (1981) presented an $O(n \log^2 n)$ algorithm for constructing the GVD for a set of n points and line-segments in the plane. Kirkpatrick (1979), proposed an $O(n \log n)$ algorithm for constructing the Voronoi diagram of an arbitrary collection of n disjoint points and open line-segments.

After the Voronoi diagram has been constructed, the robot's start and goal points are retracted to it in the following manner. Let $\text{proj}_{\text{start}}$ and $\text{proj}_{\text{goal}}$ be the configurations on the C_{free} 's boundary, which are closest to q_{start} and q_{goal} respectively. The half-line ℓ emanating from $\text{proj}_{\text{start}}$ and $\text{proj}_{\text{goal}}$ and passing through q_{start} and q_{goal} respectively intersects the $\text{GVD}(C_{\text{free}})$ in some configurations. The closest to $\text{proj}_{\text{start}}$ and $\text{proj}_{\text{goal}}$ intersections of ℓ with $\text{GVD}(C_{\text{free}})$ can then be obtained. These intersections are the two retracted points q_{start}' and q_{goal}' of q_{start} and q_{goal} respectively on $\text{GVD}(C_{\text{free}})$. Figure 3.5 illustrates this retraction. Since there are $O(n)$ edges in the $\text{GVD}(C_{\text{free}})$, this retraction can be accomplished in $O(n)$ time.

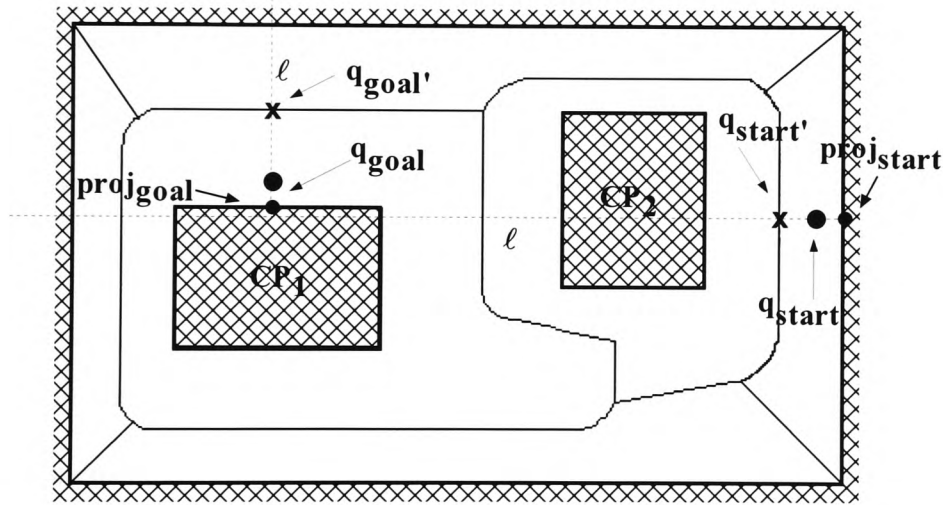


Figure 3.5 Illustration of the retraction of q_{start} and q_{goal} on the $\text{GVD}(C_{\text{free}})$.

When the roadmap ($\text{GVD}(C_{\text{free}})$) has been constructed and the robot's start and goal locations are retracted to it, it can then be searched for a path from q_{start} to q_{goal} . The collision-free path between q_{start} and q_{goal} generated by this approach, consists of three sub-paths: the straight line path from q_{start} to q_{start}' , a path in the diagram from q_{start}' to

q_{goal}' and a straight line path from q_{goal}' to q_{goal} . The search process requires $O(n)$ time, leading to an $O(n \log n)$ time for the overall approach.

The Voronoi diagram has been used widely for robot path planning, (Ò'Dúnlaing and Yap, 1985), (Ò'Dúnlaing *et al*, 1986), (Ò'Dúnlaing *et al*, 1987), (Takahashi and Schilling, 1989). Canny and Donald (1988), proposed a simplified Voronoi Diagram for the motion planning problem, which is composed only of straight line-segments and is easier to extend to higher dimensions.

The advantage of the generalised Voronoi diagram approach is that it establishes safe paths, because when the robot is moving along these paths it stays as far away as possible from the environment's obstacles. However, a major disadvantage of this approach is that in general it does not return optimal paths. As with the visibility graph the disadvantage of this method is that the construction of the Voronoi diagram in higher dimensions is not very obvious and requires considerable computational time.

3.2.3 Freeway Method

The freeway method was developed by Brooks (1983), the idea is similar to that of the Voronoi diagram, that is to keep the robot as far away as possible from the obstacles. It is supposed that the robot is a convex polygon and that the obstacles are represented as unions of convex polygons. This approach does not require the robot's configuration space to be calculated instead it uses the robot workspace W . The freeway method, in general, extracts geometric features called freeways from the workspace, connecting them using a one-dimensional network called freeway net. After the robot's start and

goal points are retracted to this network, the method searches the freeway net for a collision free path between these points.

The freeways are created in those places in the environment that are not occupied by obstacles. The environment's obstacles contribute in the creation of the freeways by letting the edges of the adjacent obstacles, which face each other produce a straight linear generalised cylinder. A freeway is a straight linear generalised cylinder, the straight axis (spine) of which is annotated with a conservative description of the robot's (R) free orientations, while its reference point R_r is moving along it. This method was originally intended to be used for two-dimensional polygonal workspaces where, the robot translates and rotates. Figure 3.6 illustrates the concept of the freeway.

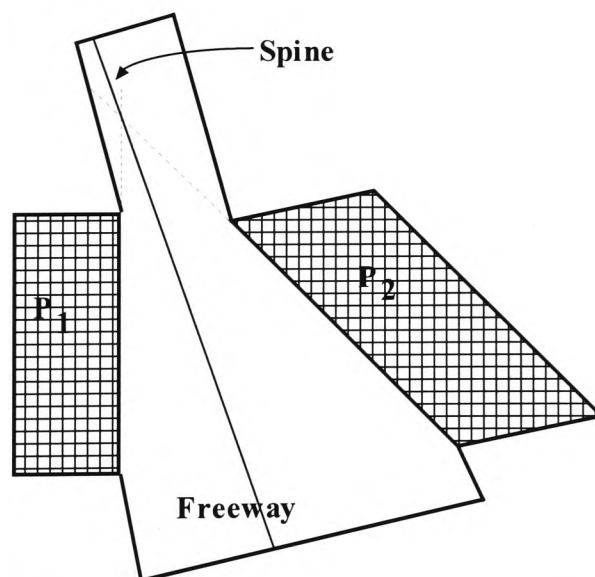


Figure 3.6 A two dimensional straight linear generalised cylinder created by the environment's obstacles P_1 and P_2 .

A two dimensional straight linear generalised cylinder was defined in (Latombe, 1991) as follows:

*'A **two-dimensional straight linear generalised cylinder** is a region of \mathbb{R}^2 obtained by sweeping a straight line-segment, the **cross-section**, along a straight line **the spine**. An origin and an orientation are defined on the spine. The cross-section stays perpendicular to the spine. It is partitioned by the spine into two segments, the right and left cross-sections. The lengths of the right and left cross-sections are independent, continuous, piecewise linear functions of the abscissa along the spine. The two lines drawn by the extremities of the cross-section are called the right and the left **sides** of the cylinder.'*

In this approach the robots obstacle-free sub-space E of the workspace W , ($E = W \setminus \bigcup_i P_i$, where P_i are the environment's obstacles) is represented as overlapping generalised cones (freeways). For the construction of the freeways all pairs of obstacles' edges are considered. With every obstacle's edge, a supporting line can be associated which includes this edge and separates the plane into two half-planes. One of these half-planes contains the obstacle and is called the inner half-plane of the edge and the other, which does not contain the obstacle, is called the outer half-plane. A pair of edges E_1, E_2 produce a generalised cylinder if the following two conditions are satisfied:

1. At least one vertex of each edge is on the outer half-plane of the other.
2. The inner product of the outward pointing normals of E_1 and E_2 is negative.

These conditions simply ensure that the edges E_1, E_2 "face" each other. The construction of a generalised cone up to this point requires $O(n^2)$ time, where n is the total number of the obstacles' edges in the scene.

The next stage of the construction is the formation of the spine of the cone. The spine is formed by a candidate pair of edges and is the bisector of the intersection of the two outer half planes defined by the edges. If the candidate edges are parallel the spine is parallel and equidistant to them, otherwise it is the bisector of the angle formed by the extension of the two edges. Each side of the cone is constructed of one of the edges (E_1 or E_2) extended at each extremity, by a half line parallel to the spine, see Figure 3.6.

The cone is then checked against all the obstacles to see if it lies in a free space. Every obstacle can be intersected with the cone in time $O(n)$, where n is the number of the obstacle's edges. If an empty intersection arises nothing has to be done, otherwise the intersection is projected to the spine of the cone and the portion of the cone (slice), which intersects with the obstacle is truncated. This procedure is called *truncation of the cylinder* and requires $O(n)$, where n is the number of the obstacle's vertices. All the disjoint slices of the cone, which do not include portions of both original edges, are then discarded. Figure 3.7 illustrates this procedure.

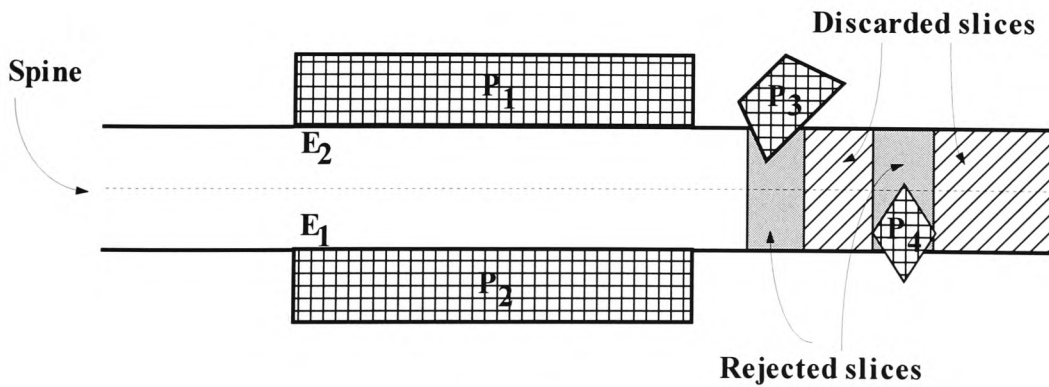


Figure 3.7 Truncation of non-free slices of a generalised cone.

The overall computational time of the representation of the free space by freeways is $O(n^3)$, where n is the total number of the edges in the polygonal obstacles.

Figure 3.8 illustrates five generalised cones generated by the obstacles of a bounded environment (the boundary of the environment is considered as an obstacle).

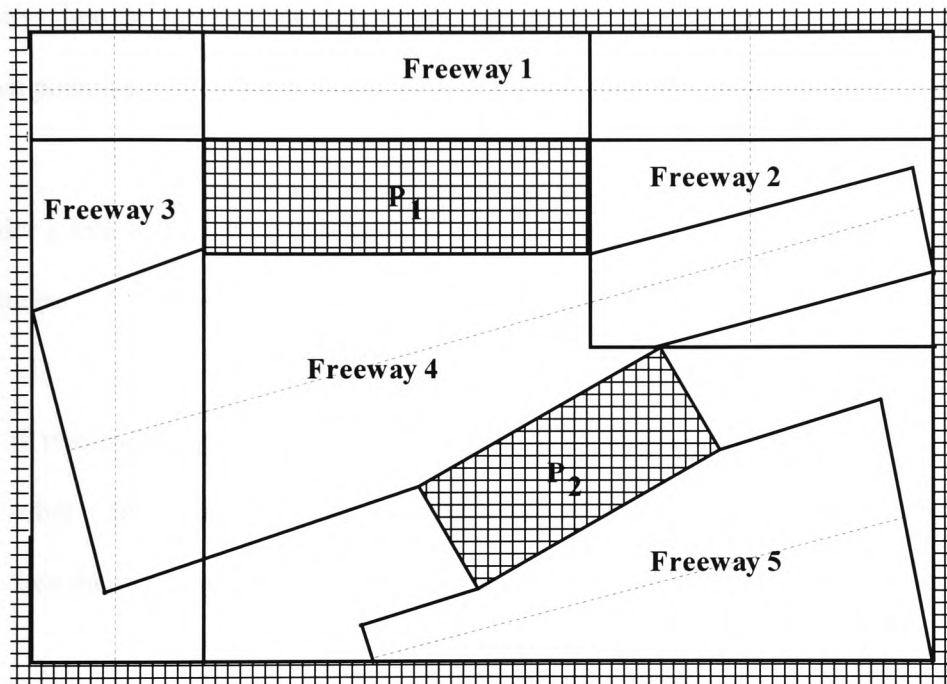


Figure 3.8 Five freeways generated by the environments boundary and the environment's obstacles.

When all the generalised cones, which represent the free space, have been constructed, all the pairs of cones are examined for possible intersections of their spines and if there are some, to identify where these intersections occur. Each spine intersection point for each cone is accompanied by a subset of $[0, 2\pi)$, which determines the orientation interval of the robot R , such that R stays inside the corresponding generalised cone.

Next, an undirected graph G is generated, which is called freeway net. Nodes of this graph are all the points that the spines of two freeways intersect and are contained in both freeways. As there are at most $O(n^2)$ cones, their pair-wise intersection has an upper bound of $O(n^4)$. If the start and the goal points are on the spine of the freeways, then they are added as nodes in the graph and their intervals are associated with these nodes in the graph. If the robot's start and goal positions do not lie on spines of freeways then the robot has to move from its start position to a spine and from a spine to its goal position.

In order a link to be created in the graph, the link must meet one of the two following conditions:

1. If two nodes are on the spine of the same freeway and the intersection of the robot's free-orientations interval associated with these nodes is a non-empty set, then there is a link between these two nodes in the freeway net.
2. If two nodes represent the same point in the x-y plane but for two different freeways and the intersection of the robot's free-orientations interval associated with these nodes is a non-empty set, then there is a link between these two nodes in the freeway net.

The first link condition corresponds to legal intra-freeway motions of the robot, while the second one corresponds to legal inter-freeway motions of the robot. The constructed freeway net is searched for a path between that initial and the goal nodes.

The freeway method is suitable for two-dimensional environments where the robot translates and rotates. This method works quite fast when it is applied to relatively uncluttered environments, but its main drawback is that it is not complete, which means that it sometimes fails to find a path for the robot even if one exists. The incompleteness of the method is due to the fact that it rests on some intuitive assumptions, such as the spine of the freeways to be the bisector of the angle that the two supporting lines of the obstacles' edges create or the fact that the robot travels on the spines.

3.2.4 Silhouette Method

The silhouette method (or roadmap algorithm) was proposed by Canny (1988) and is the only *complete general* approach for robot path planning, which runs in a single exponential time in the dimension of the configuration space. It is complete because it guarantees to find a semi-free path providing that one exists, otherwise it reports failure. It is general because it is applicable in configuration spaces with arbitrary dimensions.

In this approach, the closure of the collision-free configuration space of the robot is input as a compact semi-algebraic set. Since the method operates in compact semi-algebraic sets it is also suitable for planning paths of multiple robots or articulated arms. The silhouette method uses tools from Differential Geometry such as “stratifications” and from the Elimination Theory such as “Multivariate Resultants” in order to achieve its complexity bounds. The method will be described here using a simple example, which is taken from (Canny, 1988), independently of these tools.

Let the closure of the free space ($\text{cl}(C_{\text{free}})$) be a compact sub-set of \mathbb{R}^m , where m is dimension of C and denoted by S . In this example, $m = 3$ and S is an ellipsoid which is pierced by a cylindrical hole. This is depicted in Figure 3.9.

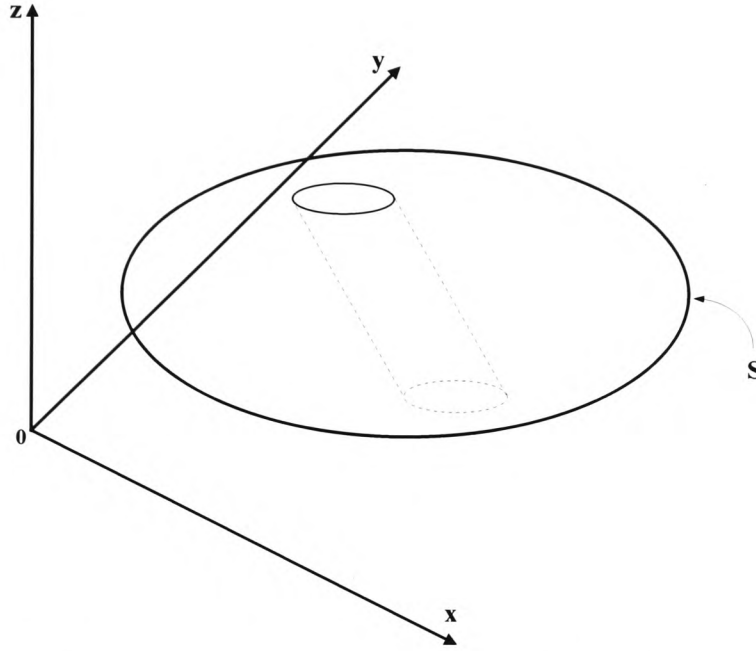


Figure 3.9 Illustration of S , which is a compact sub-set of \mathbb{R}^m .

The main idea behind the algorithm is to construct a one-dimensional roadmap $R(S)$ that is connected within each connected component of S . The connectivity of $R(S)$ can be explicitly computed and can be represented as a graph. Whenever the existence of a path between two arbitrarily chosen points, say p_1 and p_2 in S , is questioned, it can be established, by just searching the roadmap after p_1 and p_2 are connected to some nearby points of it. Therefore if the two points lie in a connected component of S , a path that connects them should exist in the roadmap $R(S)$. The roadmap $R(S)$ is constructed as follows.

At the first stage, the algorithm reduces the dimension of the problem by one by considering slices through the space S with a hyperplane P_c of dimension $m-1$. In this example, S is a three-dimensional space, so P_c is a plane. The plane P_c is taken perpendicular to one of the axes, say the x -axis (the choice of the axis is arbitrarily made) of the co-ordinate system in which S is presented. At every position $x = c$ the plane P_c defines a two-dimensional slice of S , which is $P_c \cap S$. This is illustrated in Figure 3.10.

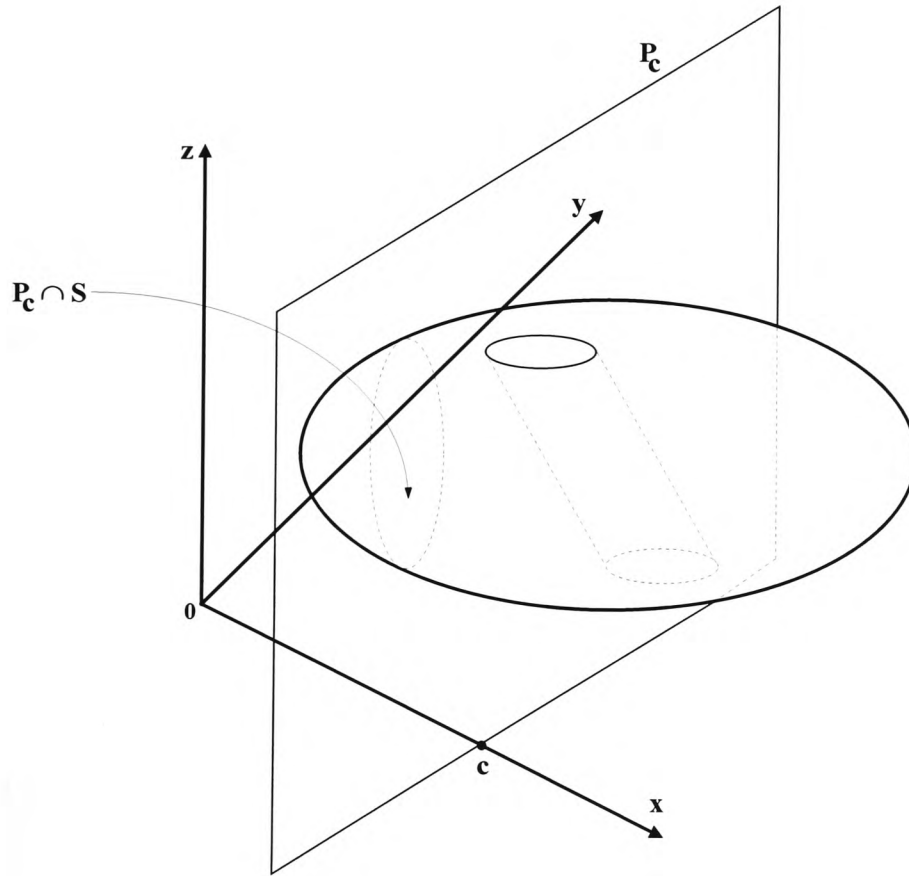


Figure 3.10 The slice $P_c \cap S$ of S when the P_c is at position $x = c$.

If a direction in the plane P_c is arbitrarily chosen, say y -direction, then $R(S)$ contains all the locally extremal points of $P_c \cap S$ in that direction. Locally extremal points are the local minima, maxima and inflection points. The fact that S is a compact set guarantees that every connected component of $P_c \cap S$ has such local extremal points. In this way, any point in $P_c \cap S$ can be connected with a path to $R(S)$. Some attention is required in the choice of co-ordinates, so that there is a finite number of extremal points in every slice (almost any choice is suitable, (Canny, 1988)).

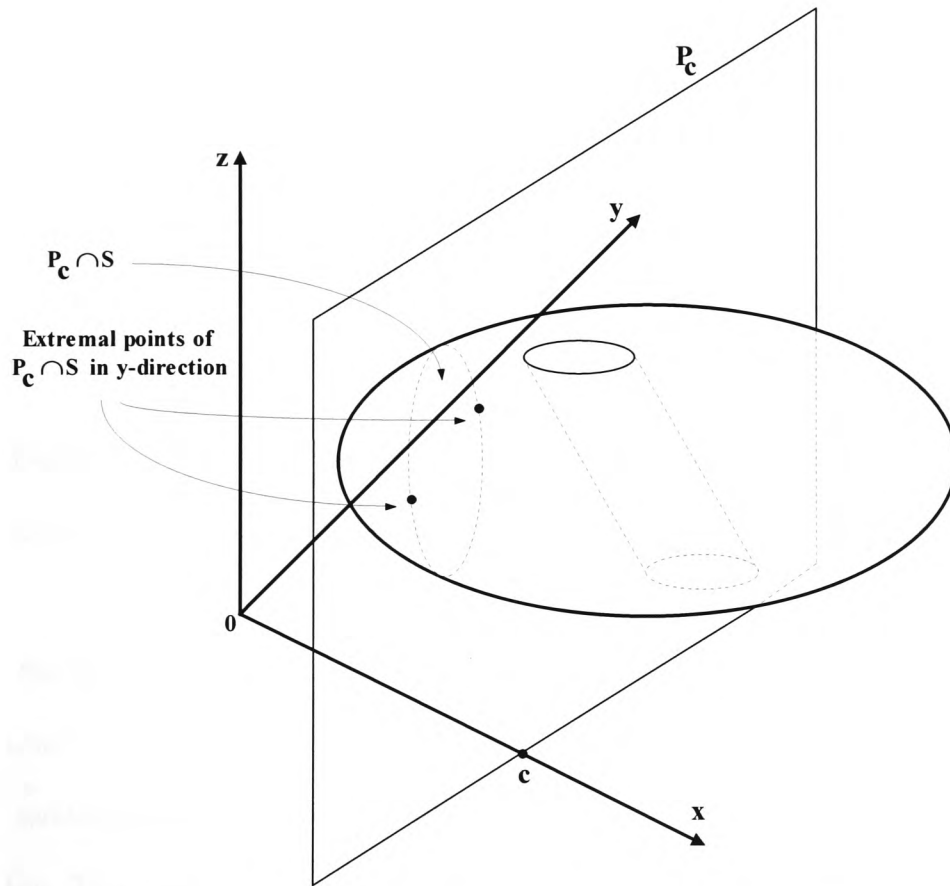


Figure 3.11 The two extremal points of the slice $P_c \cap S$ in y -direction.

The algorithm sweeps the plane P_c across the x -axis (this is when the c 's value is varied), so the extremal points trace out one-dimensional curves called silhouette curves. These curves form the spine of the skeleton $R(S)$, as is shown in Figure 3.12.

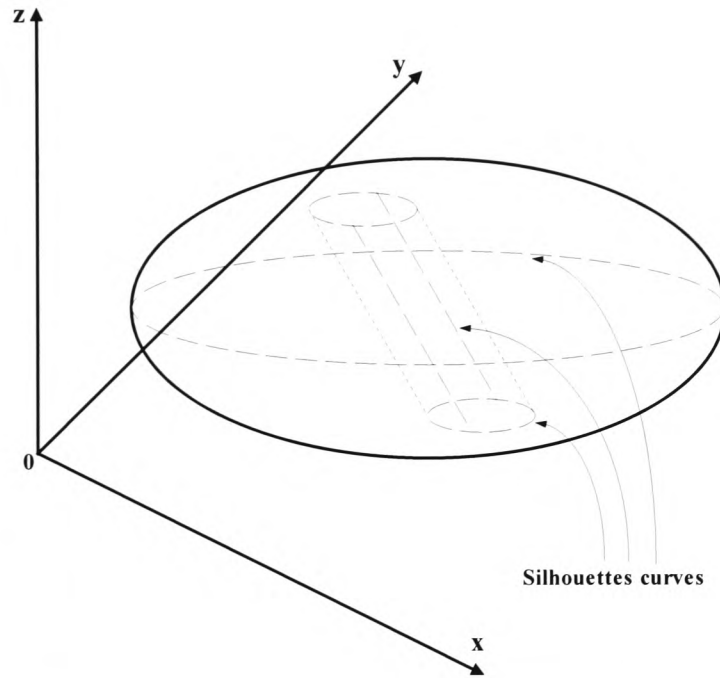


Figure 3.12 When the $P_c \cap S$ is swept across the x -axis the extremal points form silhouette curves (long-dashed curves).

What can be noticed from Figure 3.12 is that although the set S is connected, the silhouette is not. Since it is desired that the roadmap $R(S)$ represents S 's connectivity, some additional silhouette curves must be added in order to make the representation complete. This can be done during the sweep process in the following way.

As the value of c start increases from some value $c = c_0$, the extremal points of the $P_c \cap S$ slice trace out smooth curves. If the curves were connected for some c with $c_0 < c < c_1$

they would remain connected for a small increment of c by δc . However, this is not true, as it can be noticed from Figure 3.13, new extremal points appear or disappear in certain slices. This happens for a finite number of values of c , these values are called *critical values*. The new extremal points themselves are called *critical points*, Figure 3.13.

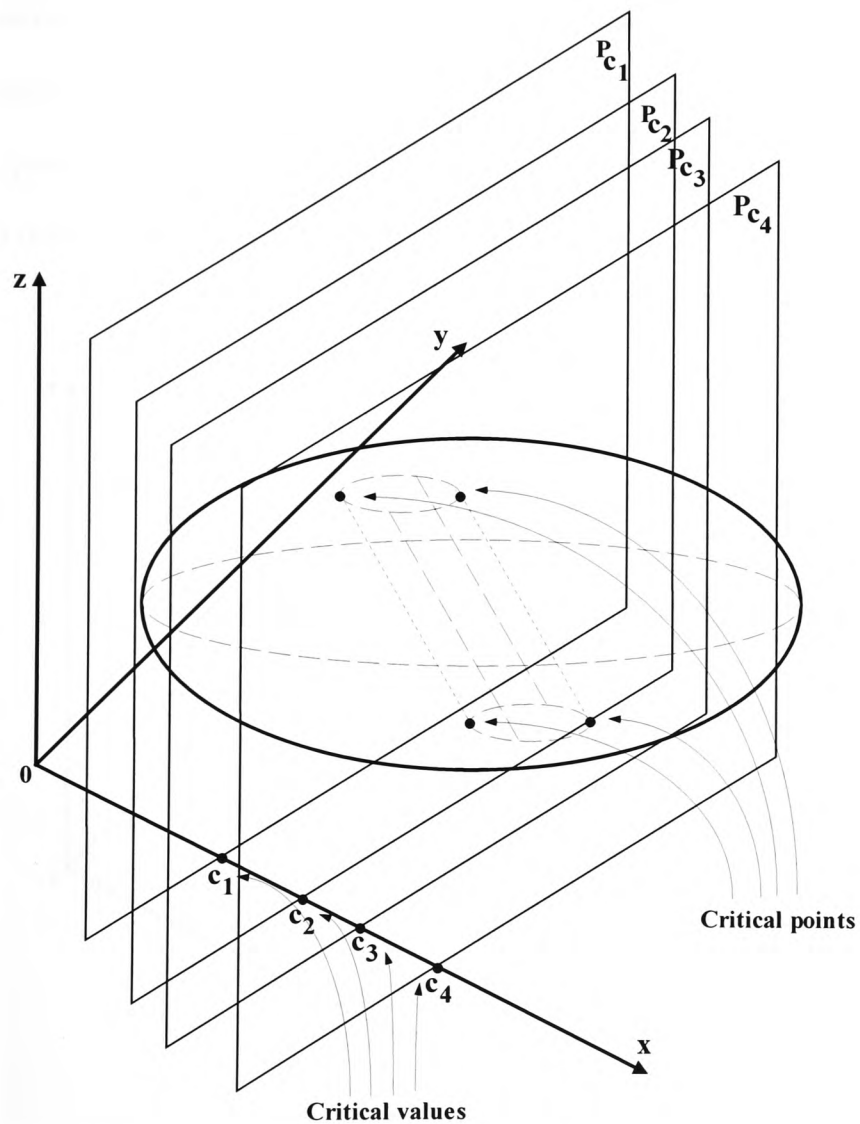


Figure 3.13 At the critical values c_i new extremal points appear in the set S .

Whenever the sweep-plane passes through a critical point the algorithm connects this point to the rest of the silhouette with a path, which lies on the $P_c \cap S$. Note that there is always such a path because of the way silhouette is defined. This process is called *linking* and is achieved by recursive callings of the roadmap algorithm on the $m-1$ -dimensional set $P_c \cap S$.

In the second recursive call of the algorithm a hyperplane of dimension $m-2$ perpendicular to, say the y -axis, will sweep the $P_c \cap S$ across the y -axis and the new extremal points in a third dimension (say z -axis direction) will form new silhouette curves, as illustrated in Figure 3.14.

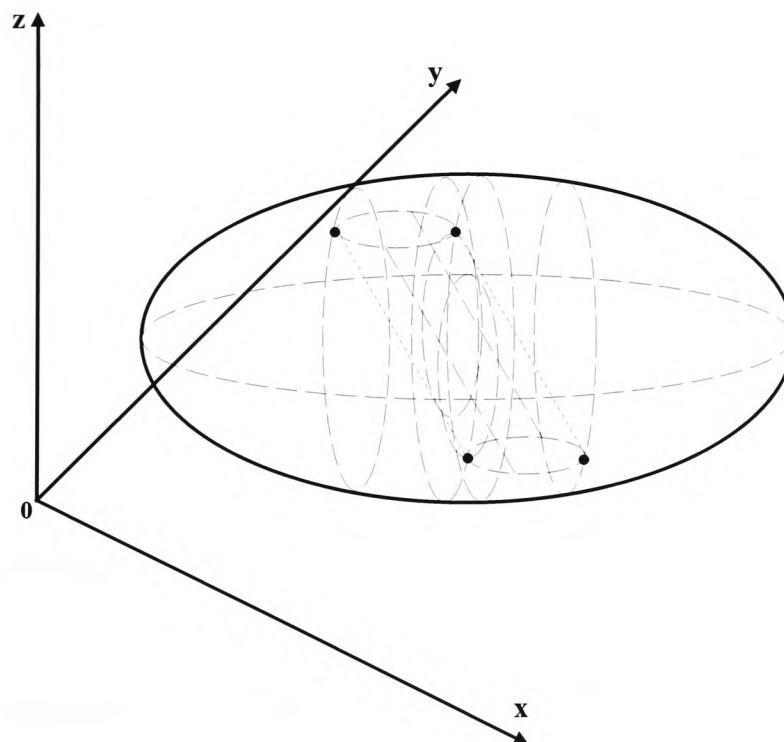


Figure 3.14 The complete silhouette of set S .

The recursion in general ends when there are not any critical points to connect to the silhouette in the set which is currently being swept, or when the dimension of the set is two. The robots start and goal point can be connected to the roadmap $R(S)$ by treating them as critical points.

This technique is the only known complete method for robot path planning which runs in a single exponential time. In particular, its running time is exponential in the dimension of the robots configuration space. The exponent of the algorithm is equal to the robots degrees of freedom. When the robot's degrees of freedom are low, the method can solve the path planning problem in polynomial time as a function of the environment's complexity.

The disadvantage of this method is that the expensive computational complexity makes it impractical. Therefore, this method is mostly used in theoretical algorithms analysing complexity rather than as a practical approach for solving the robot path planning problem. Besides the impractical nature of this approach, another disadvantage is that the path established by this method lies in the $cl(C_{free})$. Therefore, this method does not give rise to safe paths.

3.2.5 Probabilistic Roadmaps (PRM)

This method constructs a roadmap in a random fashion. The overall computation of this approach is carried out in two phases. The *pre-processing phase* (or *learning phase*) and the *query phase*.

The robot motion planning problem in general has two formulations determined by the availability of information about the robot's start and goal points a priori planning. These formulations are the *single shot problem* and *learning problem*. Single shot problems are these, where the robot's start and goal points are given in advance and the objective is to find a collision-free path between them. Learning problems are those where the robot's start and goal points are not given in advance and the objective is to construct a data structure that can be later used for any query. The single shot version of the problem is computationally cheaper to be solved, but if several different motions for a robot are requested to be computed in the same environment, it is computationally more efficient to solve the learning problem version.

The PRM approach can solve the learning problem thus it is regarded as a learning approach³. In the pre-processing phase, a roadmap which is an undirected graph $G = (V, E)$, is incrementally constructed in a probabilistic way. This construction is achieved by repeatedly generating random collision-free configurations of the robot. Every such configuration is added to V and is connected to the graph G by adding some edges to E . These configurations are connected using a very fast but not so powerful motion planner, called *local planner*. Note that every newly added configuration is connected by the local planner only to some neighbouring configurations within a distance d under some metric D . A potential field method can be used as a local planner, (Overmars and Švestka, 1995). The aim of this phase is to construct a network, which reasonably covers the C_{free} . Note that at the end of this phase, additional nodes

³ The roadmap approaches (section 3.2) and the cell decomposition approaches (section 3.3) are considered as learning approaches, while the potential field method (section 3.4) is considered as single-shot method. The potential field method is single-shot, because the goal point contributes in the construction of the potential field (see section 3.4 for details).

may have to be added in 'difficult' places of the C_{free} in order that the representation of the robot's free space is more complete.

In the query phase, a query is encountered (hence the name of the phase), such as whether there is a collision-free path between two configurations, q_{start} and q_{goal} . In this phase an attempt is made to connect these configurations to some nodes of the roadmap constructed in the pre-processing phase and then this roadmap is searched for a free path. A path between the q_{start} and the q_{goal} configurations and two nodes of the graph, say \bar{q}_{start} and the \bar{q}_{goal} respectively, can be found by using the local planner of the pre-processing phase.

This approach was proposed by (Kavraki and Latombe, 1994a), (Kavraki and Latombe, 1994b) for motion planning of manipulators with many degrees of freedom. Independently presented in (Overmars and Švestka, 1995) for solving the motion planning problem for robots with few degrees of freedom and in (Švestka and Overmars, 1997), for motion planning of both symmetrical non-holonomic car-like robots and car-like robots, that can only move forward. It was further explored by Kavraki *et al* (1996) and Barraquand *et al* (1997). A very similar idea to the probabilistic roadmap technique was independently proposed by Horsch *et al* (1994), for motion planning of manipulators with up to 12 degrees of freedom. Their approach randomly generates configurations and connects them to their nearest- k neighbours for a small k , in order to build a graph $G = (V, E)$. This task however, may result in a number of disconnected sub-graphs. A connection between these sub-graphs is then generated by reflecting randomly at the C -obstacles. A brief survey on PRM

along with a proposed PRM and some experimental results can be found in (Kavraki and Latombe, 1998).

The PRM approach is probabilistically resolution-complete. This means that it can solve the problem (find a collision-free path in the open C_{free} if one exists) with a probability approaching unity (1), providing that it is executed for a sufficient amount of time. By this it is meant that the probability that the planner will find a path bounds its computational complexity.

Experiments have shown that for the motion planning of a manipulator with 16 degrees of freedom in 'difficult' environments (these are environments with narrow passages), the learning phase requires time of the order of hundreds of seconds to adequately capture the C_{free} with a roadmap of approximately 4700 nodes. When the time allowed for the query phase is of the order of a few seconds, the success rate of the approach is over 90%. For the experiment a DEC Alpha workstation was used (Kavraki and Latombe, 1998). For the motion planning of a car-like robot in difficult environments, the success rate of the approach is over 90%, with the learning phase requiring time of the order of a few seconds and the query phase requiring time of the order a few seconds, using a Silicon Graphics Indigo workstation (Švestka and Overmars, 1997).

3.3 Cell Decomposition Approaches

The main principle of the cell decomposition methods is to decompose the robot's collision-free configuration space (C_{free}) into a finite number of non-overlapping regions

called *cells*. In exact cell decomposition, the cells are of simple shapes (usually triangles and/or trapezoidal when the C_{free} is two-dimensional and parallelepiped when the C_{free} is three-dimensional) and cover the entire C_{free} . In approximate cell decomposition, the C_{free} is decomposed in cells of predefined shape (usually squares when the C_{free} is two-dimensional and cubes when the C_{free} is three-dimensional), whose union is a conservative approximation of the C_{free} . In exact cell decomposition method the boundaries of the C_{space} 's objects are used for the generation of the cells, thus this method is *object dependent*, while in approximate cell decomposition the boundaries of the C_{space} 's objects are not used for the generation of the cells and thus this method is *object independent*. Once the C_{free} is decomposed, a *connectivity graph* is constructed, which represents the adjacency of the cells. The connectivity graph is then searched for a channel between the cells, which contain the robot's start and goal configurations respectively. If there is a channel between the cells, which contain q_{start} and q_{goal} , a path, is extracted from it. In the following section the exact and approximate cell decomposition methods will be described for solving the basic robot movers' problem. For a detailed description of the decomposition methods see, (Yap, 1987) and (Latombe, 1991).

3.3.1 Exact Cell Decomposition

Since the decomposition of the C_{free} is exact, the exact cell decomposition method is complete. The main difficulties of this approach are the selection of the geometry of the cells and the construction of the adjacency information.

Here the exact cell decomposition approach will be discussed in its simplest form (for solving the basic movers' problem). Note that the generalised approach given by Schwartz and Sharir (1983b) is far more complicated and uses the important result of Collins (1975) for deciding the satisfiability of Tarski sentences. This method will be described here very briefly.

Consider the robot's configuration space of Figure 3.15. The robot's configuration space is a bounded polygonal region, which contains four holes (C-Obstacles). Its boundary is also considered as an obstacle. The robots start and goal configurations are denoted by q_{start} and q_{goal} respectively.

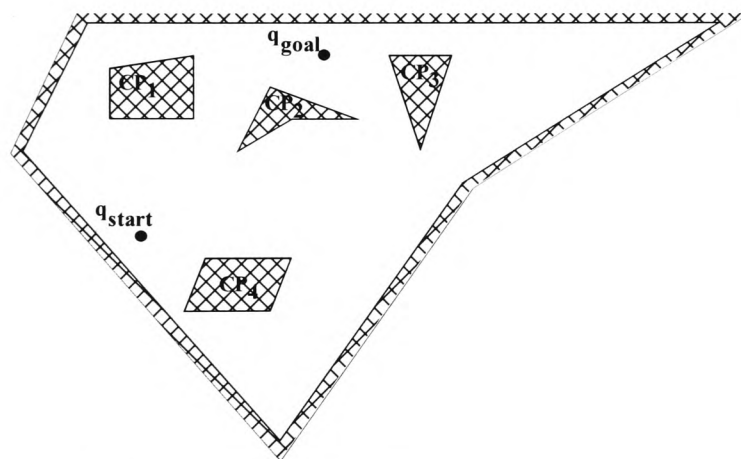


Figure 3.15 The robot's configuration space externally bounded by an obstacle and internally bounded by four obstacles.

As was mentioned in section 3.2 the first step of the exact cell decomposition approach is to decompose the C_{free} into a finite number of non-overlapping convex cells whose union is exactly the C_{free} . There are two different types of decomposition, the first is

covering and the second is *partitioning* (Keil and Sack, 1985). Covering a polygon means that the decomposition it is permitted to contain mutually overlapping pieces. Partitioning a polygon means completely dividing its interior into non-overlapping cells. Here it is of interest to decompose the C_{free} by partitioning it, because it is required that any point of C_{free} is contained in exactly one cell.

The optimal convex decomposition of a polygon (that is, decomposition into the smallest possible number of convex polygons) can be computed in polynomial time in the number of the polygons vertices (Keil and Sack, 1985). If the polygon to be decomposed contains holes, the problem of decomposing it, is NP-hard, (Lingas, 1982).

However, a decomposition method, which is non-optimal and is due to Chazelle (1987), can be used. This is called trapezoidal decomposition and it decomposes the polygonal region in trapezoids and triangles. (Note that triangulation could be used instead, to decompose the environment in triangular cells.) This method sweeps a line parallel to the y-axis across the C_{free} . Each time a vertex of a CP_i is encountered the algorithm generates at most two line-segments, which emanate from the encountered vertex and are extended to each extremity of the y-direction until they cross an edge of a CP_i . If a crossing with such an edge does not occur (this case arises when the C_{free} is not bounded) the decomposition contains cells extended infinitely. Figure 3.16 illustrates the trapezoidal decomposition of the C_{free} of Figure 3.15.

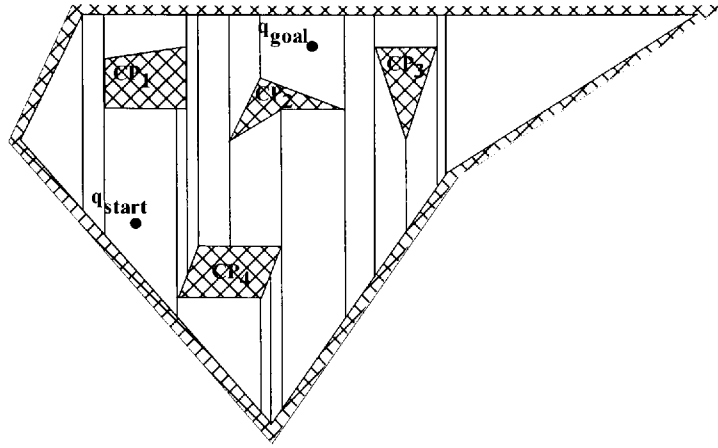


Figure 3.16 Trapezoidal decomposition of the C_{free} .

The sweeping process used to identify the CP_i 's vertices requires $O(n \log n)$ computational time, where n is the total number of the C-Obstacles' vertices. Note that while the sweep-line is sweeping across the C_{free} , the erection of the vertical line-segments, the identification of the cells, which contain the q_{start} and q_{goal} configurations and the generation of the connectivity graph G can be achieved concurrently.

The connectivity graph G is an explicit representation of the adjacency of the cells of C_{free} . G is an undirected graph whose nodes represent the cells and each edge of it represents the adjacency between two cells. Two cells are adjacent when they share an edge with length greater than zero. The total number of nodes and edges of the connectivity graph is $O(n)$. Figure 3.17 illustrates the connectivity graph, which represents the adjacency of the cells of the decomposed C_{free} of Figure 3.16.

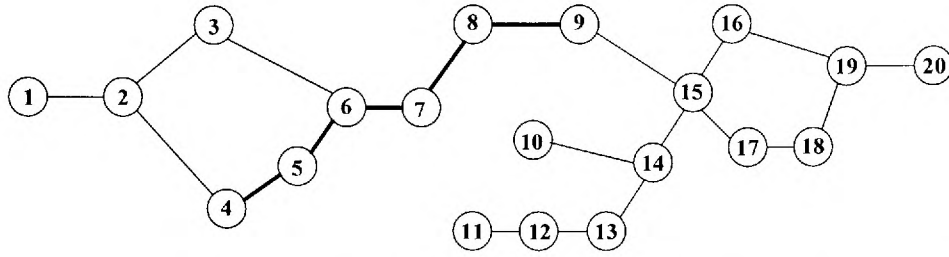


Figure 3.17 The connectivity graph G , which represents the adjacency of the cells of Figure 3.16. The bold line is the path from the cell, which contains q_{start} to the cell, which contains q_{goal} .

After the connectivity graph is generated it can be searched for a path (note this is not a real path of the robot), from the node, which corresponds to the cell that contains the q_{start} configuration to the node, which corresponds to the cell that contains the q_{goal} configuration (bold lines in Figure 3.17). This path defines the channel, which is a sequence of adjacent cells, between the cell, which contains the q_{start} configuration and the cell, which contains the q_{goal} configuration. In Figure 3.18 the shaded cells illustrate the channel.

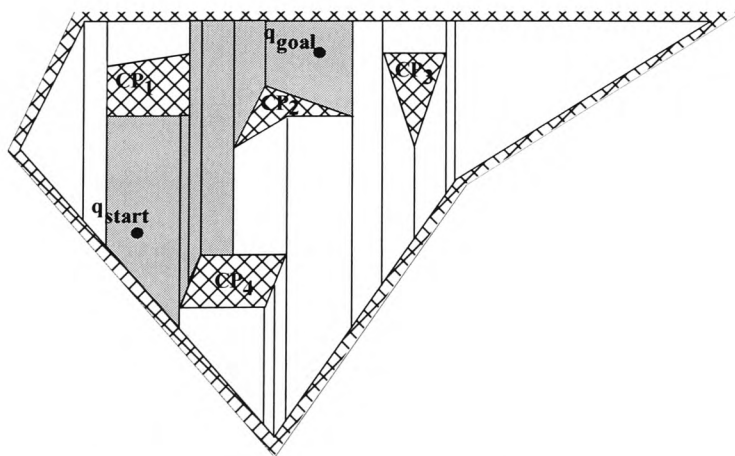


Figure 3.18 The channel (shaded cells) is a sequence of adjacent cells between the cells, which contain q_{start} , and q_{goal} .

Any search engine can be used to search the connectivity graph. A path for the robot can then be defined within the channel by connecting the midpoints of the portions of the boundary shared by each successive cell in the channel. The bold line in Figure 3.19 illustrates, the robot's path between the q_{start} and q_{goal} .

Note that heuristics (i.e. the A* algorithm) can be used to optimise the length of the robot's path. In this case a graph G^* needs to be constructed (instead of the connectivity graph) with nodes the q_{start} , the q_{goal} and all the midpoints of the portions of the common boundaries of the adjacent cells. An edge between two nodes in G^* appears if the corresponding midpoints are adjacent to the same cell. The graph G^* can then be searched using the A* algorithm to find the shortest path. Note that this path is the shortest path between q_{start} and q_{goal} among all the paths that pass through the midpoints of the common boundaries of adjacent cells and not the actual shortest path between these two points.

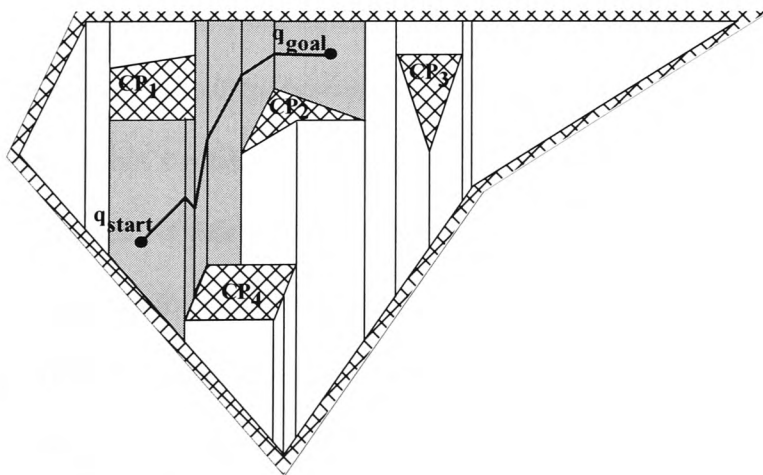


Figure 3.19 The path between q_{start} and q_{goal} .

Since the search process requires $O(n^2)$ computational time, where n is the number of vertices of the graph, the overall computational complexity of the approach is $O(n^2)$, where n is the total number of the C-obstacles' vertices.

The advantage of this technique is that the paths that it generates are most of the time safer than the ones produced by either the visibility graph or the silhouette approaches. The main disadvantage is that they are not optimal in general. This method can be extended to three-dimensional configuration spaces populated by polyhedral obstacles. In this case the decomposed cells are parallelepipeds.

An exact cell decomposition approach was proposed by Schwartz and Sharir (1983a), for planning paths for a ladder (line-segment robot) and for compact connected two-dimensional polygonal robot in an environment W populated by polygonal obstacles. The case where the robot R is a ladder will be discussed here, the solution to the path planning problem when R is a polygon as defined above is very similar. The robot R can translate freely and rotate about its one endpoint. In this approach the environment W is decomposed into *non-critical regions* bounded by *critical curves*. A critical curve is the locus of the robot's reference point while the robot is moving having a *critical contact* with an obstacle. Critical contacts can be defined in many ways some of which are: a vertex of the robot touching an edge of an obstacle, an edge of the robot touching a vertex or an edge of an obstacle or the robot touching two or more obstacles at the same time. There are five different types of contacts in total when the robot is a ladder. Every non-critical region is a subset of the x-y plane and is the maximal subset of the

positions of W at which, at least one orientation of R exists, such that R does not intersect any critical curves.

The feasible orientation for R when its reference point (one end of the ladder) lies in a non-critical region is expressed as a finite union of open sets of angles. The bounds of each set correspond to angles at which the robot makes contact with some obstacles. The robot can move from one point to another in the same non-critical region if the orientation of the robot at each point belongs to the same open set. The robot can move from one non-critical region to another if and only if, their boundaries share an open portion β of a critical curve and for every point $p \in \beta$ the intersection of the robot's feasible orientations of the two regions is a non-empty set.

Once W is decomposed into cells (non-critical regions), a connectivity graph is generated, which represents the adjacency of the cells. This graph is then searched for a channel from the robot's start point to its goal point and a path is finally defined within this channel. Since the contacts between the robot and the obstacles define the critical curves, there are in total $O(n^2)$ (actually this bound is due to some more complicate curves such as the *conchoid of Nicomedes*, obtained by particular type of contacts) critical curves and since the critical curves can intersect each other there are in total $O(n^4)$ critical curve sections, leading to a $O(n^5)$ overall time, where n is the complexity of the obstacles.

An optimised variant of the above algorithm was proposed in (Leven and Sharir, 1985) for motion planning of a ladder in a two-dimensional space populated by polygonal

barriers (obstacles). The algorithm's computational complexity is $O(n^2 \log n)$, where n is the total number of the obstacles' vertices.

A general exact cell decomposition algorithm was proposed by Schwartz and Sharir (1983b). Since this algorithm is exact its complexity depends on the complexity of the free space, which in turn depends on the complexity of the number of multiple contacts between the robot and the obstacles. The approach proposed by Schwartz and Sharir can be used for the motion planning of a robot with an arbitrary number of degrees of freedom. The only restriction of this method is that both the robot and the obstacles should be described as semi-algebraic sets. This approach is quite complicated to implement and its computational complexity is $O(n^{2^{d+6}})$, where n is the total number of obstacles' vertices and d is the total number of the degrees of freedom of the robot. This result is practical only for configurations with low dimensions and robots with small number of degrees of freedom. Quoting the authors '*The approach is catastrophically inefficient*', i.e. $O(n^{4096})$ for a 6 dof robot. Again, this method serves as proof for the decidability of the general path planning problem.

A variant of the exact cell decomposition was proposed by Avnaim *et al* (1988), for solving the two-dimensional instance of the movers' problem, when the robot rotates as well as translates. In their approach, they first constructed a graph G_1 , in which every node is associated with a face of the C_{free} (a C-surface patch) and for every pair of adjacent faces an edge connects the corresponding nodes in the graph. Note that in this case $C = \mathbb{R}^2 \times S^1$. In the next step a triangular decomposition T_{start} of $C_{\text{free}_{\theta_{\text{start}}}}$

(respectively T_{goal} of $C_{\text{free}_{\text{goal}}}$), where $C_{\text{free}_{\text{start}}}$ is a connected component of the $\text{cl}(C_{\text{free}}) \cap \text{cl}(C_{\text{free}_{\text{goal}}})$, (respectively $C_{\text{free}_{\text{goal}}}$ is a connected component of the $\text{cl}(C_{\text{free}}) \cap \text{cl}(C_{\text{free}_{\text{start}}})$) are produced. A graph G_2 is then constructed in which every node is associated with a cell in T_{start} and for every pair of adjacent cells an edge connects the corresponding nodes in the graph. Similarly, a graph G_3 is constructed for the T_{goal} decomposition. The last step of their algorithm is to merge the graphs G_1 , G_2 and G_3 to form a graph G' , by connecting every node of graph G_1 (these nodes correspond to faces of ∂C_{free}) to every node of the graphs G_2 and G_3 whose corresponding faces and cells share a straight line-segment of non-zero length. Finally G' is searched for a path between the cells which contains the start and goal points. The computational complexity of their algorithm is $O(K \log K + F)$, where K is the sum of the number of edges which compose the boundary of $\text{cl}(C_{\text{free}}) \cap \text{cl}(C_{\text{free}_{\text{start}}})$ and $\text{cl}(C_{\text{free}}) \cap \text{cl}(C_{\text{free}_{\text{goal}}})$, which in the worst case is $O(m^2 n^2)$ and F is the number of the faces of the ∂C_{free} . m and n are the number of the vertices of the robot and the obstacles respectively.

When the number of the robot's degrees of freedom is small, less powerful but more practical approaches using different approximation schemes and heuristics have been proposed by, (Brooks, 1983) see also section 3.2.3, (Lozano-Pérez, 1981) see also section 3.3.2, (Guibas *et al*, 1989) and (Seneviratne *et al*, 1997).

3.3.2 Approximate Cell Decomposition

As with the exact cell decomposition, this approach decomposes the C_{free} into a finite number of cells and a path is found through a channel of cells. In the approximate cell decomposition, the cells have a pre-specified simple shape. Therefore, the union of the cells approximates the C_{free} hence the name of the method. The size of the cells can be locally adapted by the geometry of the C-Obstacles. This approach was first proposed by Lozano-Pérez (1981) and was further developed and used by, (Brooks and Lozano-Pérez, 1983), (Faverjon, 1984), (Laugier and Germain, 1985), (Faverjon, 1986), (Kambhampati and Davies, 1986), (Noborio *et al*, 1990), (Zhu and Latombe, 1991), (Barbehenn and Hutchinson, 1995) and (Katevas *et al*, 1998).

Since the approach is conservative about the approximation of the C_{free} , it is not complete therefore it can fail to find a path even if one exists. However, it is attractive because cells are generated by iterating the same simple computation and in general, the method is fast and easy to implement. The method will be described here for solving the basic movers' problem.

Consider the robot's configuration space of Figure 3.20. Without loss of the generality it is assumed that the robot's configuration space is bounded by a square, (the choice of the shape is made to make the exposition of the method simpler) and populated by three polygonal C-Obstacles. Note that the boundary of the robot's configuration space is considered as C-Obstacle as well. The start and goal configurations of the robot are the q_{start} and q_{goal} respectively.

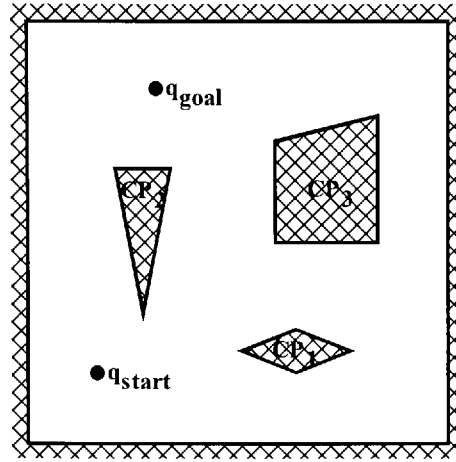


Figure 3.20 Two-dimensional configuration space.

This method recursively decomposes the configuration space into smaller rectangloid (rectangloid decomposition) called cells. In every recursive call of the decomposition over a cell, four new identical rectangloids are generated. This decomposition can be represented by a tree of degree four, therefore it is called quadtree decomposition (Samet, 1980), (Samet, 1990). If R is a rectangloid $R \subset \mathbb{R}^m$ which bounds the C_{free} , then a rectangloid decomposition K of the space R is a finite set of rectangloids k_j , $j = 1..p$ such that:

$$R = \bigcup_{j=1}^p k_j \quad (3.1)$$

and

$$\forall f, h \in [1..p] : \text{int}(k_f) \cap \text{int}(k_h) = \emptyset \quad (3.2)$$

Two cells are adjacent when they share a set in \mathbb{R}^{m-1} of non-zero length measurement. In this example, since the configuration space C is two-dimensional the rectangloids will be of dimension two (which are rectangles) and in particular they will be squares.

A classification of the cells k_j (rectangloids in \mathbb{R}^m), which reflects the occupancy of the C-obstacles within the cells can be defined as follows:

- $\forall j \in [1..p], \forall i \in [1..n], \text{int}(k_j) \cap \text{CP}_i = \emptyset : k_j \text{ is EMPTY}$
- $\forall j \in [1..p], \forall i \in [1..n], \text{int}(k_j) \subseteq \text{CP}_i : k_j \text{ is FULL}$
- $\forall j \in [1..p], \forall i \in [1..n], (\text{int}(k_j) \cap \text{CP}_i \neq \emptyset) \wedge (\text{int}(k_j) \not\subseteq \text{CP}_i) : k_j \text{ is MIXED}$

Figure 3.21 illustrates the successive decomposition of R (environment of Figure 3.20) after two recursive calls of the decomposition procedure. The first decomposition decomposes R into four identical squares separated by the bold lines in Figure 3.21. The second decomposition decomposes R into sixteen identical squares separated by plain lines in Figure 3.21. The white cells are the EMPTY cells and the grey cells are the MIXED cells. At every call of the decomposition, only cells, which lie entirely in the C_{free} or in a CP_i , are not further decomposed.

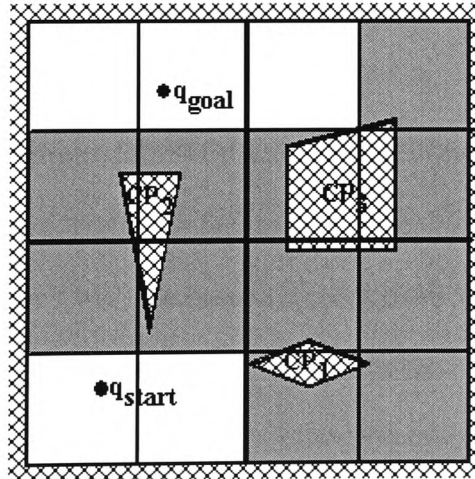


Figure 3.21 The cells obtained after two successive decompositions of the environment.

After each decomposition of the space R into cells, a connectivity graph G is generated representing the adjacency of the cells. Vertices of this graph are all the EMPTY and MIXED cells. An edge occurs in the graph between two vertices if their corresponding cells are adjacent. Since the environment is two-dimensional, two cells are adjacent when they share an edge with length greater than zero. Figure 3.22 illustrates the connectivity graph, which represents the adjacency of the cells of the decomposed C_{free} of Figure 3.21.

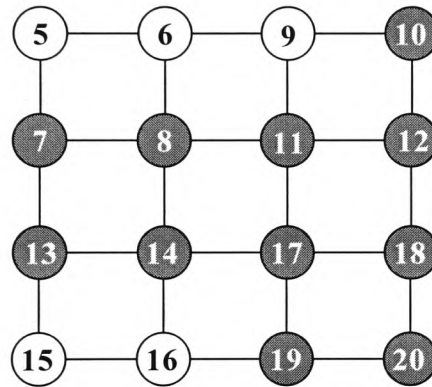


Figure 3.22 The connectivity graph G , representing the adjacency of the cells of Figure 3.21.

A channel is a sequence of adjacent EMPTY and/or MIXED cells. If a channel contains only EMPTY cells it is called, EMPTY-channel otherwise it is called MIXED-channel. A path, which lies within an EMPTY-channel is a collision-free path. A path, which lies within a MIXED-channel, is not always collision-free path.

A hierarchical path planning method is defined by starting with a coarse decomposition of the configuration space and searching the corresponding connectivity graph for an

EMPTY-channel between the cells, which contain the robot's q_{start} and q_{goal} configurations. If such a channel does not appear the algorithm then recursively decomposes the MIXED cells until an EMPTY-channel is obtained or a predefined cell-resolution is achieved.

As it can be seen from the connectivity graph of Figure 3.22, there is not an EMPTY-channel connecting the cells, which contain the q_{start} and q_{goal} configurations (these are nodes six and fifteen respectively). Therefore, the space R is further decomposed. More specifically only the MIXED cells are further decomposed. Figure 3.23 illustrates the decomposed space after the third successive decomposition of the space R .

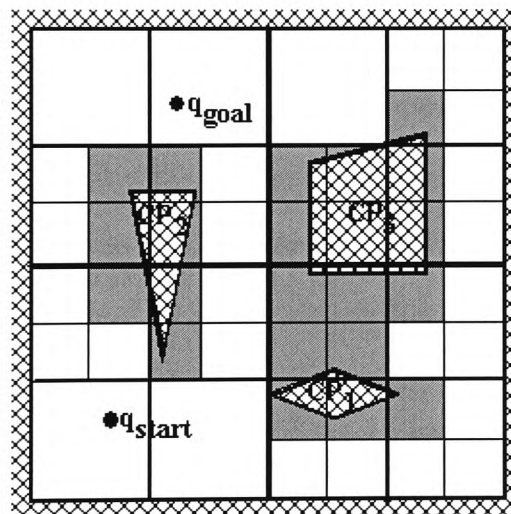


Figure 3.23 Cells obtained after the third decompositions of R .

The connectivity Graph G of the environment of Figure 3.23 is illustrated in Figure 3.24.

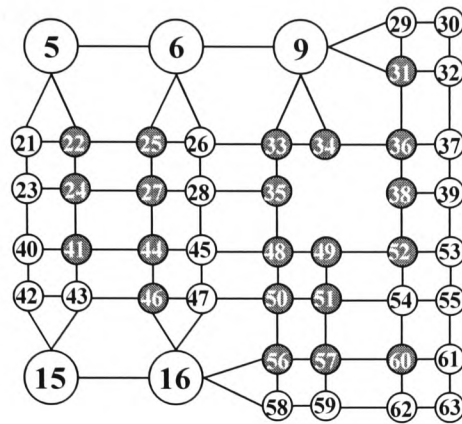


Figure 3.24 The connectivity graph G , representing the adjacency of the cells of Figure 3.23.

In Figure 3.24 the third successive decomposition of R gives rise to an EMPTY-channel between the cells, which contains the q_{start} and q_{goal} configurations (nodes six and fifteen respectively). Once an EMPTY-channel is found, a path can then be extracted from it, by connecting the q_{start} and q_{goal} configurations with a polygonal line passing through midpoints of the portions of the boundary shared by each successive cell in the channel. This is indicated by a dotted line in Figure 3.25.

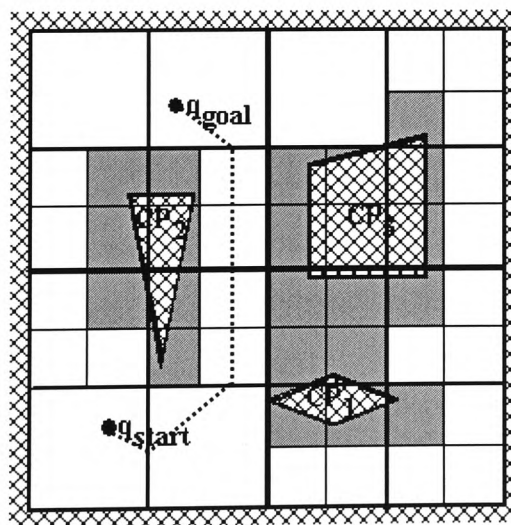


Figure 3.25 The path (dotted line), between q_{start} and q_{goal} .

For the case when the common midpoints of the portions of the boundary shared by each successive cell in the channel are subsets of the edge/face of the same cell, additional vertices in the path should be introduced, such as configuration Q in Figure 3.26.

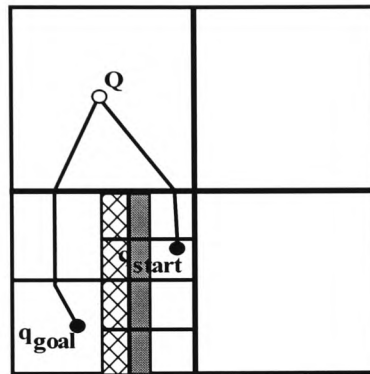


Figure 3.26 Illustration of a case where additional vertices should be introduced.

For the search of the connectivity graph G in every recursive step of the algorithm, heuristics can be used to speed up the procedure of finding an EMPTY-channel. Note that sometimes even though an EMPTY-channel exists a somewhat shorter (distance-wise) path could be found by further decomposing the MIXED cells and possibly give rise to other EMPTY-channels, obviously this is at the expense of computational time.

The approximate cell decomposition is not a complete method because the union of the decomposed cells is an approximation of the C_{free} . Therefore, the method might fail to find a path even if one exists. However, the precision of the approximation can be

appropriately tuned (at the expense of computational time) to make the approximation very small, hence this method is resolution-complete.

An advantage of the approximate cell decomposition is that it is straightforward and is relatively easy to implement. Also since the size of the decomposed cells can be controlled, the method can give rise to a desired clearance of the path, which in turn can allow small perturbations on the robot's motion (due to control errors) while the robot is moving along the path.

Since its computational time grows with respect the size of the configuration space, the method is practical for configurations with low dimensions (less than five (Latombe, 1991)). When the environment is three-dimensional, the decomposition is called octree, because it can be represented by a tree of degree eight. In general, the decomposition of a configuration space of dimension m can be represented by a 2^m -tree, which is a tree of 2^m degree.

3.4 Potential Field Approach

The idea behind the potential field method is somewhat different than the idea behind all the methods discussed so far, which construct a network of one-dimensional curves in the robot's configuration space or physical space and then search this network for a path between the robot's start and goal points. In the potential field method, the concept of electrical potential fields is used, as heuristics to guide the search for a path between the robot's start and goal points in its physical space. The potential fields were first

introduced by Khatib and Mampey in 1978 as an obstacle avoidance technique, (Hwang and Ahuja, 1992a). The method was used and further developed for robot path planning by (Khatib, 1986). Boissiere and Harrigan (1988) used the potential field method for local collision avoidance in a human Tele-operated puma. When the robot was due to collide with an obstacle while moving along the course given by the operator the PUMA reacted to the repulsive potential of the obstacle and changed its path. Tilove (1990), presented an overview of the artificial potential field method for path planning, described variations of the method and compared the performance of different algorithms. Hwang and Ahuja (1992b), presented a path planning algorithm based on the potential fields. In their approach path planning was done at two levels. At the first level a global planner generates a graph-like representation of the free space between minimum potential valleys (MPV) and also defines a path and the orientation of the robot along this path such that the chance for collision is minimum. At the second level a local planner moves the robot along the path found by the global planner and alters the path or the robot's orientation, if there is a need to, in order to avoid collisions. If the local planner fails at any point the global planner is used to generate new path and orientations. The process is repeated until a path is found or there are no more paths left for further examination. Juang (1998), used potential fields for real-time collision avoidance for an industrial manipulator. He presented an algorithm for fast calculation of the distance between the manipulator's links and the environment's obstacles and used this distance to create artificial repulsive forces between the robot and the obstacles. A collision avoidance control scheme based on the potential fields was then presented. A historical review of the potential field method for path planning and control can be found in (Koditschek, 1989).

In the potential field method, the robot is considered as an electric particle moving under the influence of an artificial potential field U in a configuration-like space. Hence, the local variations of the potential field reflect the topology of robot's free space. A uniform artificial attractive potential is defined over the goal point, which attracts the particle (robot) towards it and an artificial repulsive potential is defined over the obstacles, which repel the particle from them. The motion of the particle is locally generated by a potential function, which combines both of the attractive and repulsive components of the field. An artificial force can then be produced at a current point as the negated gradient of the total potential field, which can be used to move the robot towards the most promising direction.

The obstacles in the environment are not represented by geometrically volumes but by potential functions, therefore the only knowledge that is required by the algorithm is local. Thus this method is suitable for real-time obstacle avoidance or/and on-line path planning. Note that this technique was initially intended for obstacle avoidance and not for path planning.

A typical potential function is defined as the summation of an attractive potential and a repulsive potential. The former pulls the robot towards the goal while the latter pushes the robot away from the obstacles. The force \vec{F} produced by potential field U and applied on the particle at any point q is equal to the negated gradient vector of the potential field U . More formally when the robot's workspace is two-dimensional, the gradient vector of U at any point $q = (x, y)$ can be defined by the equation 3.3.

$$\vec{\nabla} U = \begin{pmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{pmatrix} \quad (3.3)$$

The potential field function should attain its minimum at the goal point and its maximum at the obstacles. In all other points, of the robot's free space the function should slope towards the goal point. Figure 3.27 illustrates a two dimensional workspace and its corresponding combined attractive and repulsive potential fields.

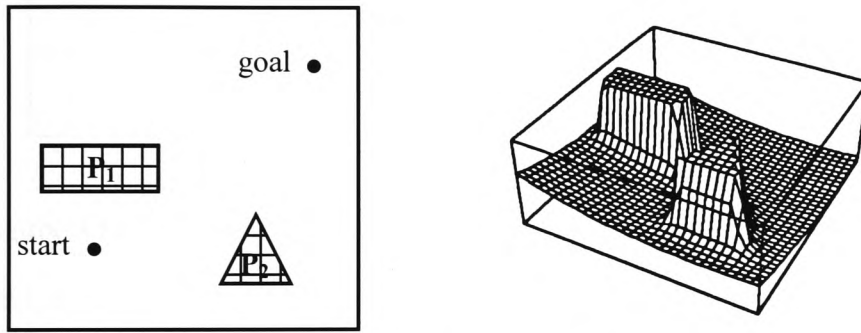


Figure 3.27 The summation of the artificial attractive and repulsive potential fields of a two-dimensional environment.

To model the environment involves determining field functions for the robot's goal point and the obstacles. An attractive potential field associated with the goal point is defined as a parabolic attractor, for example by the equation 3.4.

$$U_{attr}(q) = \alpha \cdot d(q, q_{goal})^2 \quad (3.4)$$

Where α is a positive scaling factor and $d(q, q_{\text{goal}}) = \|q - q_{\text{goal}}\|$. Note that U_{attr} function over the goal point is equal to zero. The force \vec{F}_{attr} applied on the particle by the attractive potential field U_{attr} , when is at a point q is, $\vec{F}_{\text{attr}} = -\vec{\nabla} U_{\text{attr}}(q)$. Figure 3.28 illustrates the parabolic well created by the artificial attractive potential field.

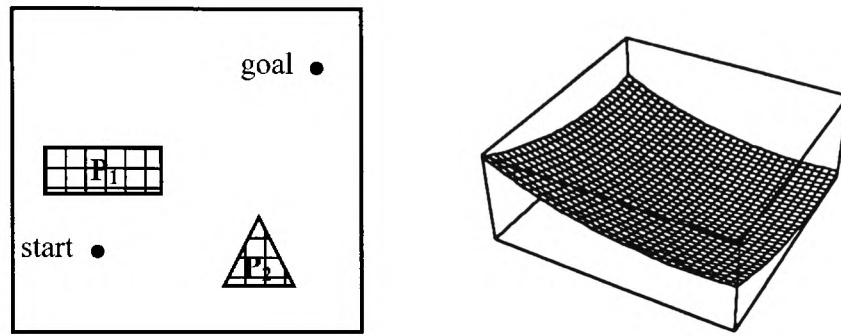


Figure 3.28 The parabolic well created by the artificial attractive potential field of the goal point in a two-dimensional environment.

The repulsive force of an obstacle is modelled as a potential barrier that is erected to infinite as the robot approaches the obstacle. A repulsive potential field associated with every obstacle of the environment can be defined by the equation 3.5.

$$U_{\text{rep}}(q) = \begin{cases} \beta (d_{\min}(q, P_i)^{-1} - \rho_0^{-1}) & \text{if } \rho(q) \leq \rho_0 \\ 0 & \text{if } \rho(q) > \rho_0 \end{cases} \quad (3.5)$$

Where β is a positive scaling factor, $d_{\min}(q, P_i)$ is the minimum distance from q to a P_i obstacle and ρ_0 is positive constant, which describes the distance of influence of the obstacles. The function of the repulsive potential is piecewise, because it is desired that the particle is not influenced by the repulsive potential, when it is far away from the obstacle P_i . The force \vec{F}_{rep} applied on the particle by the repulsive potential field U_{rep} when is at any point q is, $\vec{F}_{rep}(q) = -\vec{\nabla} U_{rep}(q)$. Figure 3.29 illustrates the high value of the potential in the obstacles, attained by the artificial repulsive potential field.

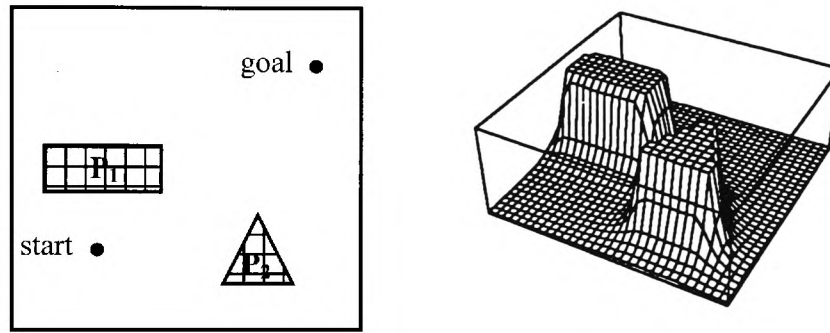


Figure 3.29 The artificial repulsive potential field of the obstacles in a two-dimensional environment.

A problem that can arise when a P_i is a non-convex polygon is that it is possible for the particle to oscillate between the adjacent sides of a reflex vertex of the obstacle, due to the simultaneous influence of a force field on the particle by the two sides. A way around this problem is to consider every non-convex obstacle in the environment as a concatenation of a finite number of non-overlapping convex sets. The repulsive potential of the obstacle is then equal to the sum of the repulsive potential of each convex set.

The main disadvantage of the potential field approach is that the robot can be trapped in local minima of the potential function and hence this method does not always find a path from q_{start} to q_{goal} even when one exists. This fact makes the method incomplete. A solution for the local minima problem has been proposed by Khosla and Volpe (1988), by defining a potential function with only one minimum or in the worst case with a few. Rimon and Koditschek 1992, proposed a local minimum-free function called, global navigation function for robot path planning and control, in Euclidean spaces when the obstacles of the environment are spherical or star-shaped. An alternative is to escape from local minima and to begin the search again (see section 3.4.1 for details).

When the attractive and repulsive potentials are defined, a path can be found from the robot's start point to its goal point iteratively, by moving the robot towards the most promising direction defined by the artificial force \vec{F} induced by the potential function and proceeds in that manner by some increment until it reaches the goal point. This path corresponds to the path traced by a small ball placed at the start point and let roll on the potential surface until it reaches the goal point (Cameron, 1994). Suppose that the surface of the potential net is viscous and the ball will not overpass the goal point.

Two common approaches that can be used for the generation of a path using the artificial potential field are the *depth first search technique without backtracking* and the *best first search* (Latombe, 1991). With depth first search technique without backtracking, the constructed path is constituted by successive line-segments starting from the robots start point. Every segment is computed at a point derived from the

previous line-segment and is oriented along the negated gradient of the potential at this point. The main disadvantage of this technique is that because the robot follows the deepest fall of the potential function until it reaches the goal point, it is difficult to handle a situation when the robot is trapped in a local minimum before it reaches the goal point.

An alternative is a best-first search technique. This technique uses a fine grid over the space, it starts from the robot's start point and iteratively constructs a tree whose leaves are points of the grid, each of them with a pointer to its parent node in the tree. The leaves, which correspond to the most promising potential-wise points of the grid, are further expanded until the goal point is retained. If the goal point is attained the path is obtained by backtracking all the pointers in the tree from the goal point to the start point. If the goal point is not reached and the entire grid has been examined then there is no path between the robot's start and goal points. Otherwise, the robot is trapped in a local minimum. In this case the algorithm 'fills' the well of the minimum until it reaches a saddle point and resume the search along the negated gradient until the robot reaches the goal (Barraquand and Latombe, 1991).

3.4.1 Randomised Path Planning

The randomised path planner (RPP) was proposed by Barraquand and Latombe (1990), and is one of the most effective potential field based planners. Their approach is very like the best first search. The planner starts from the robot's start point and follows the deepest fall of the potential function until it reaches a minimum. If this minimum is the global (the goal point) then a path between the robot's start and goal point is obtained.

If this is a local minimum then a random walk mechanism is activated to move the robot to some other point and then the deepest fall strategy is resumed until a minimum is reached again. Note that the random walk is a simulation of the Brownian motion that influences the natural particles. If the newly obtained minimum is not the same as that previously encountered then the new motion is added to the path. In this way a graph with nodes which are all the local minima of the potential function is created. The planner searches the environment until the goal point is encountered or until it gives up.

3.5 Discussion

In this chapter the robot motion planning problem was considered and some robot motion planning techniques have been presented. The robot motion planning problem has been shown to be a very hard problem especially in its full generalisation. This justifies the existence of so many different approaches to tackle the problem and the increased research efforts consumed on the subject in the last twenty years.

The majority of the robot motion planning techniques discussed in this chapter are concerned with solving the basic movers' problem, the reason for this is twofold. Firstly because it is easier to expose the methods by solving a simple instance of the robot motion planning problem than by solving the problem in its full generality. Secondly, because this thesis is concerned with the motion planning of autonomously guided vehicles, which in general operate in two-dimensional physical spaces and they do not have configuration spaces of high dimensionality.

It has been shown that complete planners are computationally extremely expensive and they mostly serve as proof of the decidability of the general movers' problem rather than for practical use in the real world, especially when the instance of the problem being tackled is complicated. On the other hand many heuristic approaches, resolution-complete approaches and probabilistically resolution-competent approaches have been proposed to make the problem more tractable and its solution more pragmatic and practical but of course at the expense of the solution's completeness. In (Barraquand *et al*, 1997), it was stated that no tailor-made planner is likely to be the most efficient for all possible problems and that every application requires a hand made solution. For instance the PRM and the Voronoi diagram can provide good solutions for the robot motion planning problem in two-dimensional static environments but when the environment contains moving obstacles both approaches may fail to generate solutions.

In the subsequent chapters of the thesis the path planning problem for an AGV in two-dimensional static and dynamic environments is considered. Specifically in chapter four the static problem is encountered and an algorithm for solving it is proposed. The algorithm is based on the concept of visibility graph. The reason for this is that the visibility graph approach works quite fast in configurations of two-dimensions and in general establishes optimal paths, which is always desired. The proposed algorithm is called V*MECHA and it finds the shortest semi-free path (using Euclidean metric) for an AGV in a two-dimensional static environment by constructing a reduced visibility graph. The V*MECHA algorithm is based on the V*GRAPH algorithm and has been designed to correct and overcome the latter's deficiencies.

In chapter five the V*MECHA algorithm is extended to be applicable in dynamic environments. An algorithm is proposed which is called D*MECHA and finds the time-minimal motion for an AGV between two query points. In the rest of the thesis various extension of the D*MECHA algorithm are proposed in order to increase its applicability.

Every algorithm proposed in the thesis is accompanied with an empirical analysis of its computational time and space, a proof of its correctness and a proof of its optimality.

3.6 References

- AKMAN, V. 1987. Unobstructed paths in Polyhedral Environments. *Lecture notes in Computer Science*, **251**. Berlin: Springer - Verlag.
- ALEXOPOULOS, C. and GRIFFIN, P. 1992. Path Planning for a Mobile Robot. *IEEE Transactions on Systems, Man and Cybernetics*, **22** (2), pp. 318 – 323.
- ASANO, T., ASANO, T., GUIBAS, L., HERSHBERGER, J. and IMAI, H. 1985. Visibility-Polygon Search and Euclidean Shortest Paths. *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*. Portland, Oregon. pp. 155 - 164.
- AURENHAMMER, F. 1991. Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys*, **23** (3), pp. 345 - 405.

- AVNAIM, F., BOISSONNAT, J. D. and FAVERJON, B. 1988. *A Practical Motion Planning Algorithm for Polygonal Objects Amidst Polygonal Obstacles*. Research Report N° 890. INRIA. Sophia Antipolis, France.
- BARBEHENN, M. and HUTCHINSON, S. 1995. Efficient Search and Hierarchical Motion planning by Dynamically maintaining Single - Source Shortest Paths Trees. *IEEE Transactions on Robotics and Automation*, **11** (2), pp. 198 - 214.
- BARRAQUAND, J., KAVRAKI, L., LATOMBE, J. C., LI, T. Y., MOTWANI, R. and RAGHAVAN, P. 1997. A random Sampling Scheme for Path Planning. *International Journal of Robotics Research*, **16** (6), pp. 759 – 774.
- BARRAQUAND, J. and LATOMBE, J. C. 1990. A Monte - Carlo Algorithm for Path Planning with many Degrees of Freedom. *Proceedings of the IEEE International Conference of Robotics and Automation*. Cincinnati, Ohio. pp. 1712 - 1717.
- BARRAQUAND, J. and LATOMBE, J. C. 1991. Robot Motion Planning: A Distributed Representation Approach. *The International Journal of Robotics Research*, **10** (6), pp.628 – 649.
- BOISSIERE, P. T. and HARRIGAN, R. W. 1988. Telerobotic Operation of Conventional Robot Manipulators. *Proceedings of the IEEE International Conference on Robotics and Automation*. Philadelphia, PA. pp. 576 – 583.

- BROOKS, R. A. 1983. Solving the Find Path Problem by Good Representation of Free Space. *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-13** (3), pp. 190 – 197.
- BROOKS, R. A. and LOZANO- PÉREZ, T. 1983. A Subdivision Algorithm in Configuration Space for Findpath with Rotation. *Proceedings of the 8th International Conference on Artificial Intelligence*. Karlsruhe, FRG. pp. 799 - 806.
- CAMERON, S. 1994. Obstacle Avoidance and Path Planning. *Industrial Robot*, **21** (5), pp. 9 - 14.
- CANNY, J. F. 1988. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA.
- CANNY, J. and DONALD, B. 1988. Simplified Voronoi Diagrams. *Discrete & Computational Geometry*, **3**, pp. 219 - 236.
- CHAZELLE, B. 1987. Approximation and Decomposition of Shapes. In: SCHWARTZ, J. T. and YAP, C. K. *Advances in Robotics: Algorithmic and Geometric Aspects of Robotics*. Volume 1. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers, pp. 145 - 185.

- COLLINS, G. E. 1975. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. *Lecture Notes in Computer Science*, **33**, pp. 134 - 183. Berlin: Springer - Verlag.
- CONN, R. A., ELENES, J. and KAM, M. 1997. A Counterexample to the Alexopoulos - Griffin Path Planning Algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, Part B, **27** (4), 721 – 723.
- DE BERG, M., VAN KREVELD, M., OVERMARS, M. and SCHWARZKOPF, O. 1997. *Computational Geometry, Algorithms and Applications*. Berlin: Springer and Verlag.
- DRYSDALE, R. L. 1979. *Generalised Voronoi Diagrams and Geometric Searching*. Technical Report STAN-CS-79-705, Computer Science Department, Stanford University, Stanford, California.
- EDELSBRUNNER, H. 1987. *Algorithms in Combinatorial Geometry*. EATCS Monographs on Theoretical Computer Science. Volume 10. Springer – Verlag.
- FAVERJON, B. 1984. Obstacle Avoidance using an Octree in the Configuration Space of a Manipulator. *Proceedings of the IEEE International Conference on Robotics and Automation*. Atlanta, USA. pp. 504 - 512.

- FAVERJON, B. 1986. Object Level Programming of Industrial Robots. *Proceedings of the IEEE International Conference on Robotics and Automation*. San Francisco, USA. pp. 1406 - 1412.
- FORTUNE, S. 1986. A Sweepline Algorithm for Voronoi Diagrams. *Proceedings of the 2nd Annual ACM Symposium on Computational Geometry*. pp. 313 – 322.
- GHOSH, S. K. and MOUNT D. M. 1987. An Output Sensitive Algorithm for Computing Visibility Graphs. *Proceedings of the 28th IEEE Symposium on Foundation of Computer Science*. Los Angeles, USA. pp. 11 - 19.
- GUIBAS, L. J., SHARIR, M. and SIFRONY, S. 1989. On the General Motion - Planning Problem with Two Degrees of Freedom. *Discrete & Computational Geometry*, **4**, pp. 491 - 521.
- HART, P. E., NILSSON, N. J. and RAPHAEL B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions of Systems Science and Cybernetics*, **4** (2), pp. 100-107.
- HORSCH, T., SCHWARZ, F. and TOLLE, H. 1994. Motion Planning with Many Degrees of Freedom - Random Reflections at C-Space Obstacles. *Proceeding of the IEEE International Conference of Robotics and Automation*. San Diego, USA. pp. 3318 - 3323.

- HWANG, Y. K. and AHUJA, N. 1992a. Gross Motion Planning - A Survey. *ACM Computing Surveys*, **24** (3), pp. 219 - 291.
- HWANG, Y. K. and AHUJA, N. 1992b. A Potential Field Approach to Path Planning. *IEEE Transactions on Robotics and Automation*, **8** (1), pp. 23 - 32.
- JIANG, K, SENEVIRATNE, L. D. and EARLES, S. W. E. 1996. Three – Dimensional Shortest Path Planning in the Presence of Polyhedral Obstacles. *Proceedings ImechE, Part C: Journal of Mechanical Engineering Science*, **210** (4), pp. 373 – 381.
- JUANG, J. C. 1998. Collision Avoidance Using Potential Fields. *Industrial Robot*, **25** (6), pp. 408 – 415.
- KAMBHAMBATI, S and DAVIS, L. 1986. Multiresolution Path Planning for Mobile Robots. *IEEE Journal of Robotics and Automation*, **RA-2** (3), pp. 135 - 145.
- KATEVAS, N. I., TSAFESTAS, S. G. and PNEVMATIKOS, C. G. 1998. The Approximate Cell Decomposition with Local Node Refinement Global Path Planning Method: Path Nodes Refinement and Curve Parametric Interpolation. *Journal of Intelligent and Robotic Systems*, **22**, pp. 289 - 314.

- KAVRAKI, L. and LATOMBE, J. C. 1994a. Randimized Preprocessing of Configuration Space for Fast path Planning. *Proceedings of the IEEE International Conference on Robotics and Automation*. San Diego, USA. pp. 2138 - 2145.
- KAVRAKI, L. and LATOMBE, J. C. 1994b. Randimized Preprocessing of Configuration Space for Path Planning: Articulated Robots. *Proceedings of the IEEE International Conference on Intelligent Robots and systems*. München, Germany. pp. 1764 - 1771.
- KAVRAKI, L. E. and LATOMBE, J. C. 1998. *Probabilistic Roadmaps for Robot Path Planning*. In: GUPTA, K. and DEL POBIL, A. P. *Practical Motion Planning in Robotics, Current Approaches and Future Directions*. Chichester, UK: John Wiley & Sons Ltd. pp. 33 - 53.
- KAVRAKI, L., ŠVESTKA, P., LATOMBE, J. C. and OVERMARS, M. H. 1996. Probabilistic Roadmaps for Path planning on High - Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, **12** (4), pp. 566 - 580.
- KEIL, M. J and SACK, J. R. 1985. *Minimum Decompositions on Polygonal Objects*. Computational Geometry, G. T. TOUSSAINT (Editor). Elsevier Science Publishers.

- KHATIB, O. 1986. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research*, **5** (1), pp. 90 - 98.
- KHOSLA, P. and VOLPE, R. 1988. Superquadric Artificial Potentials for Obstacle Avoidance and Approach. *Proceedings of the IEEE International Conference on Robotics and Automation*. Philadelphia, PA. pp. 1778 -- 1784.
- KIRKPATRICK, D. G. 1979. Efficient Computation of Continuous Skeletons. *Proceedings of the 20th Symposium on Foundations of computer Science*, pp. 18 - 27.
- KODITSCHKEK, D. E. 1989. *Robot planning and Control Via Potential Functions*. In: KHATIB, O., CRAIG, J. and LOZANO-PÉREZ, T. *The Robotics Review 1*. Cambridge, Massachusetts: The MIT Press, pp. 349 - 367.
- LATOMBE, J. C. 1991. *Robot Motion Planning*. Massachusetts: Kluwer Academic Publishers.
- LATOMBE, J. C. 1999. Motion planning: A Journey of Robots, Molecules, Digital Actors, and other Artifacts. *The International Journal of Robotics Research*, **18** (11), pp. 1119 - 1128.

- LAUGIER, C. and GERMAIN, F. 1985. An Adaptive Collision - Free Trajectory Planner. *Proceedings of the International Conference on Advanced Robotics*. Tokyo, Japan. pp. 33 - 41.
- LAUMOND, J. P. 1987. Obstacle Growing in a Polygonal World. *Information Processing Letters*, **25** (1), pp. 41 - 50.
- LEE, D. T. 1978. *Proximity and Reachability in the Plane*. Ph.D. Thesis, Coordinated Science Laboratory, University of Illinois. Urbana, IL.
- LEE, D. T. and DRYSDALE, R. L. 1981. Generalisation of Voronoi Diagrams in the Plane. *Society for Industrial and Applied Mathematics*, **10** (1), pp. 73 - 87.
- LEVEN, D. AND SHARIR, M. 1985. An Efficient and Simple Motion Planning Algorithm for a Ladder in Two-Dimensional Space Amidst Polygonal Barriers. *Proceedings of the 1st ACM Symposium on Computational Geometry*. pp. 221 – 227.
- LEVEN, D. AND SHARIR, M. 1987. Intersection and Proximity Problems and Voronoi Diagram. In: SCHWARTZ, J. T. and YAP, C. K. *Advances in Robotics: Algorithmic and Geometric Aspects of Robotics*. Volume 1. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers, pp. 187 - 228.

- LINGAS, A. 1982. The Power of Non - Rectilinear Holes. *Proceedings of the 9th Colloquium on Automata, Languages and programming*. Aarhus. pp. 369 - 383.
- LIU, Y. and ARIMOTO, S. 1991. Proposal of Tangent Graph and Extended Tangent Graph for Path Planning of Mobile Robots. *Proceeding of the 1991 IEEE International Conference on Robotics and Automation*. Sacramento - California. pp. 312 - 317.
- LIU, Y. and ARIMOTO, S. 1992. Path Planning Using a Tangent Graph for Mobile Robots Among Polygonal and Curved Obstacles. *The International Journal of Robotics Research*, **11** (14), pp. 376 - 382.
- LIU, Y. and ARIMOTO, S. 1995. Finding the Shortest Path of a Disc Among Polygonal Obstacles Using a Radius-Independent Graph. *IEEE Transactions on Robotics and Automation*, **11** (5), pp. 682 - 691.
- LOZANO-PÉREZ, T. and WESLEY, M. A. 1979. An Algorithm for Planning Collision-Free Paths Among polyhedral Obstacles. *Communications of the ACM*, **22** (10), pp. 560 - 570.
- LOZANO-PÉREZ, T. 1981. Automatic Planning of Manipulator Transfer Movements. *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-11** (10), pp. 681 - 698

- NILSSON, N. J. 1969. A Mobile Automaton: an Application of Artificial Intelligence Techniques. *Proceedings of the 1st International Joint Conference on Artificial Intelligence*. Washington D.C. pp. 509 - 520.
- NOBORIO, H., NANIWA, T. and ARIMOTO, S. 1990. A Quadtree - Based Path Planning Algorithm for a Mobile Robot. *Journal of Robotic Systems*, **7** (4), pp. 555 - 574.
- Ò'DÚNLAING, C., SHARIR, M. and YAP, C. K. 1986, Generalised Voronoi Diagrams for a Ladder. I: Topological Considerations. *Communications on Pure and applied Mathematics*, **XXXIX**, pp. 423 – 483.
- Ò'DÚNLAING, C., SHARIR, M. and YAP, C. K. 1987, Generalised Voronoi Diagrams for a Ladder: II. Efficient Construction of the diagram. *Algorithmica*, **2**, pp. 27 - 59.
- Ò'DÚNLAING, C. and YAP, C. K. 1985. A "Retraction" Method for Planning the Motion of a Disk. *Journal of Algorithms*, **6**, pp. 104 – 111.
- OKABE, A, BOOTS, B., SUGIHARA K. and CHIN, S. N. 2000. *Spatial Tessellations Concepts and Applications of Voronoi Diagrams*. Edition 2nd. Chichester: John Wiley & Sons.

- OVERMARS, M. H. and ŠVESTKA, P. 1995. A Probabilistic Approach to Motion Planning. In: GOLDBERG, K., HALPERIN, D., LATOMBE, J. C. and WILSON, R. *Algorithmic Foundations of Robotics*. Massachusetts, USA: A. K. Peters, pp. 19 - 37.
- PAPADIMITRIOU, H. C. 1985. An Algorithm for the Shortest-Path Motion in Three-Dimensions. *Information Processing Letters*, **20**, pp 259 - 263.
- PREPARATA, F. P. and SHAMOS, M. I. 1985. *Computational Geometry: An Introduction*. New York: Springer-Verlag.
- RIMON, E. and KODITSCHKEK, D. E. 1992. Exact Robot Navigation using Artificial Potential Functions. *IEEE Transactions on Robotics and Automation*, **8** (5), pp. 501 - 518.
- ROHNERT, H. 1986. Shortest Paths in the Plane with Convex Polygonal Obstacles. *Information Processing Letters*, **23**, pp. 71 - 76.
- ROHNERT, H. 1988. Time and Space Efficient Algorithms for Shortest Paths Between Convex Obstacles. *Information Processing Letters*, **27**, pp. 175 - 179.
- ROWAT, P. F. 1979. *Representing Spatial Experience and Solving Spatial Problems in a Simulated Robot Environment*. Ph.D. Thesis. University of British Columbia, Vancouver, B. C., Canada.

- SAMET, H. 1980. *Region Representation: Quadtrees from Boundary Codes*. Communications of the ACM, **23** (3), pp. 163 - 170.
- SAMET, H. 1990. *The Design and Analysis of Spatial Data Structures*. Reading (Mass.): Addison-Wesley.
- SCHWARTZ, J. T. and SHARIR, M. 1983a. On the "Piano Movers'" Problem. I. The Case if a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers. *Communications on Pure and Applied Mathematics*, **XXXVI**, pp. 345 – 398.
- SCHWARTZ, J. T. and SHARIR, M. 1983b. On the "Piano Movers'" Problem. II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds. *Advances in Applied Mathematics*, **4**, pp. 298 – 351.
- SCHWARTZ, J. T. and SHARIR, M. 1988. A Survey of Motion Planning and Related Geometric Algorithms. *Artificial Intelligence*, **37**, pp. 157 - 169.
- SENEVIRATNE, L. D., KO, W. S. and EARLES, S. W. E. 1997. Triangulation-Based Path Planning for a Mobile Robot. *Proceedings ImechE, Part C: Journal of Mechanical Engineering Science*, 211 (5), pp. 365 - 371.
- SHARIR, M. 1995. Robot Motion Planning. *Communications on Pure and Applied Mathematics*, **XLVIII**, pp. 1173 - 1186.

- ŠVESTKA, P. and OVERMARS, M. H. 1997. Motion planning for Carlike Robots Using a Probabilistic Learning Approach. *The International Journal of Robotics Research*, **16** (2), pp. 119 - 143.
- TAKAHASHI, O. and SCHILLING, R. J. 1989. Motion Planning in a Plane Using Generalised Voronoi Diagrams. *IEEE Transactions on Robotics and Automation*, **5** (2), pp. 143 - 150.
- TILOVE, R. B. 1990. Local Obstacle Avoidance for Mobile Robots Based on the Method of Artificial Potentials. *Proceedings of the IEEE International Conference on Robotics and Automation*. Cincinnati, OH. pp. 566 – 571.
- WAGER, M. L. 2000. Making Roadmaps Using Voronoi Diagrams. [WWW]. <http://www.cs.uwa.edu.au/~michaelw/hons/roadmap.html> (16 November 2000).
- WELZL, E. 1985. Constructing the Visibility Graph for n-Line Segments in $O(n^2)$ Time. *Information Processing Letters*, **20**, pp 167 - 171.
- YAP, C. K. 1987. Algorithmic Motion Planning. In: SCHWARTZ, J. T. and YAP, C. K. *Advances in Robotics: Algorithmic and Geometric Aspects of Robotics*. Volume 1. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers, pp. 95 - 143.

ZHU, D and LATOMBE, J. C. 1991. New Heuristic Algorithms for Efficient Hierarchical Path Planning. *IEEE Transactions on Robotics and Automation*, 7 (1), pp. 9 - 20.

4

The V*MECHA Algorithm for Path Planning of an AGV

A mighty maze! but not without a plan

ALEXANDER POPE
AN ESSAY ON MAN, EPISTLE 1

4.1 Introduction

In this chapter the basic movers' problem is addressed and an algorithm for solving it is proposed. However, before the problem is addressed an accurate description of its specification, will be defined as follows.

Problem 1

Consider a two-dimensional environment W populated by a finite number of two-dimensional polygonal obstacles, the AGV's start point and its goal point. The obstacles are static simple polygons and the AGV is a simple polygon, which can translate freely without rotation. The problem is to determine a safe path for the AGV between its start and goal locations providing that the shapes and locations of the obstacles, the shape of the AGV and the locations of the AGV's start and goal points are accurately known a priori planning.

It is assumed that the obstacles' configuration space can be easily constructed using the Minkowski sums (see section 2.4.4) and the AGV can be considered as a point, which translates in its configuration space. Therefore it is not peremptoriness to consider a more simplified version of the above problem in which the AGV is a point robot with strictly two-degrees of freedom (the two translations) operating in an environment W . Therefore Problem 1 can be reduced to Problem 1'.

Problem 1'

Consider a two-dimensional environment W populated by a finite number of two-dimensional polygonal obstacles, the AGV's start point and its goal point. The obstacles are static simple polygons and the AGV is a point-robot, which can translate freely, strictly¹ without rotation. The problem is to determine a safe path for the AGV between its start and goal locations providing that the shapes and locations of the

¹ Since the AGV is a point robot, it should not make any difference if it rotates or not, because is a non-dimensioned object. However, in Problem 1' the AGV is considered as a point-robot only and only because the transformation of Problem 1 to Problem 1' is possible by constructing the AGV's configuration space, which is a two-dimensional space in which the AGV can only translate.

obstacles and the locations of the AGV's start and goal points are accurately known a priori planning.

The solution to Problem 1' is adequate for solving Problem 1. The proposed algorithm for solving Problem 1' is called V*MECHA and is a roadmap approach based on the concept of the visibility graph. A visibility graph is constructed to capture the connectivity of the AGV's semi-free space and then this graph is searched to establish whether there is a semi-free path between the AGV's start location and its goal location. Thus the pure topological problem of defining a one-dimensional curve connecting two points in a space is transformed to a combinatorial one and specifically to a graph search problem.

As was mentioned in section 3.5 the V*MECHA algorithm is based on the V*GRAPH algorithm of Alexopoulos and Griffin (1992). The V*MECHA algorithm is proposed to overcome some deficiencies that arise in the V*GRAPH algorithm and solve the basic movers' problem efficiently and effectively. In section 4.2 other similar approaches to the V*MECHA algorithm for solving the basic movers' problem will be discussed. In section 4.3 the V*GRAPH algorithm will be presented. The weak points and the deficiencies of the V*GRAPH algorithm are identified in sections 4.4 and 4.5 and in section 4.6 the proposed algorithm is discussed, which is shown to overcome these deficiencies. In section 4.7 the V*MECHA algorithm is used to solve a problem and in sections 4.8 and 4.9 the admissibility and the optimality of the algorithm are demonstrated.

4.2 Related Work

As was mentioned in section 3.2.1, Nilsson (1969), was the first to consider the path planning problem for a robotic system with planning capabilities (the Shakey) in 1969. A model of the environment was used and the visibility graph method was developed to plan the robot's motions. Lozano-Pérez and Wesley (1979), further developed the method by computing the robot's configuration space and constructing a visibility graph within this space. After the visibility graph was constructed, it was searched for a path between the AGV's start and goal configurations. This algorithm was called VGRAPH and as was mentioned in section 3.2.1, it is suspected that it is in $O(n^3)$ computational time and $O(n^2)$ space, where n is the total number of the C-Obstacles' vertices.

Lee (1978), proposed an $O(n^2 \log n)$ algorithm for the construction of the visibility graph, where n is the total number of the polygonal obstacles' vertices, leading to an overall computational complexity of the visibility graph approach $O(n^2 \log n)$, where n is the obstacle's vertices. Better asymptotic upper bounds for the construction of the visibility graph were presented by (Welzl, 1985), (Asano *et al*, 1985) and (Edelsbrunner, 1987). In fact their algorithms construct the visibility graph of a polygonal scene in time $O(n^2)$, where n is the total number of the scene's vertices, leading to an overall computational complexity of the visibility graph approach in $O(n^2)$, where n is the total number of the scene's vertices.

Rohnert (1986), proposed a path planning algorithm for a point-robot operating in a two-dimensional environment populated by convex polygonal obstacles. This algorithm describes a Reduced Visibility Graph (RVG), which has smaller number of

edges than the ordinary visibility graph. This reduction was based on the observation that the shortest semi-free path between two query points, runs via edges of the obstacles and the common tangent-segments of each pair of obstacles. Note that a common tangent-segment between two convex obstacles, lies between the two vertices of contact of the tangents with the polygons and there are four in total. Rohnert gave an algorithm for the construction of the reduced visibility graph in $O(n + f^2 \log n)$ computational time and $O(n + f^2)$ space, where n is the total number of the obstacles' vertices and f is the total number of the obstacles. He then used an implementation of the Dijkstra's shortest path algorithm for a graph $G = (V, E)$ with vertex set V and edge set E , which is in $O(|E| + |V| \log |V|)$ time described by Fredman and Tarjan (1984), to compute the shortest semi-path between the AGV's start and goal points, in $O(f^2 + n \log n)$ computational time. In (Rohnert, 1988) the space complexity was further improved to $O(n)$, but at the expense of the running time of the algorithm, which was increased to $O(f n \log n)$, where n is the total number of the obstacles' vertices and f is the total number of the obstacles. These results are summarised in (Fleischer *et al*, 1992).

In (Liu and Arimoto, 1992), an algorithm for constructing a reduced visibility graph called the *Tangent graph (T-graph)*, for path planning of a point-robot in a two-dimensional environment populated by both convex and non-convex obstacles was proposed. The tangent graph is a graph whose vertices are the obstacles' vertices as well as the robot's start and goal points. The edges of this graph are the obstacles' edges as well as the edges connecting all the mutually visible vertices and define common tangents to the obstacles.. Their algorithm computes the tangent graph in $O(n(n + t) + m^2 t)$ computational time and $O(m^2 + n)$ space, where t is the total number of the

obstacles vertices, m is the number of the obstacles' convex components and n is the number of the obstacles' convex vertices. In (Liu and Arimoto, 1991) and (Liu and Arimoto, 1995), an Extended Tangent Graph (ETG), was proposed for path planning of mobile circular robots (or disks) among general polygonal obstacles. The ETG is a radius independent data structure, which registers collision-free tangents of the obstacles according to the radius of different robots. In that way re-computation of the C-space when the radius r of the robots changes is avoided, by giving a threshold interval $[l, h]$ to the edges of ETG, such that its edges are collision-free if and only if $l < r < h$.

Jiang *et al* (1996) and Jiang *et al* (1999) proposed algorithms based on the tangent graph for path planning of a non-holonomic AGV by executing reversal and forward manoeuvres. In their approach they first constructed a tangent graph in the AGV's workspace W for finding the shortest path for a point. Then they evaluate this path in order to decide whether it can be used as a reference to build up a path for the AGV. If the shortest path for a point could not be used as a feasible path of the AGV, it was then discarded and the next shortest path was considered. Finally configurations were laid sequentially on the selected path in a way that the AGV can manoeuvre from the one configuration to the next one without colliding with the environment's obstacles.

In section 4.6 an algorithm called V*MECHA is proposed for solving the basic movers' problem. The proposed algorithm has the same data structure as the visibility graph but constructs a reduced visibility graph in terms of number of edges, which in turn makes the search process for the shortest semi-path more efficient, since this process is

strongly influenced by the number of graph edges. The V*MECHA algorithm finds the shortest semi-free path between two query points in a two-dimensional environment populated by both convex and non-convex simple polygonal obstacles.

4.3 A Description of the V*GRAPH Algorithm

The V*GRAPH algorithm was developed by Alexopoulos and Griffin (1992), to find the shortest semi-free path between two query points for a point-robot (the AGV), which translates freely without rotation, in a two dimensional environment populated by simple polygonal obstacles. The V*GRAPH algorithm is very similar to the VGRAPH proposed by Lozano-Pérez and Wesley (1979). The only differences are that the latter, first constructs the entire visibility graph and then searches it for a path using the A* algorithm (Hart *et al*, 1968), while the former one interweaves the two processes. Another major difference is that the V*GRAPH algorithm does not construct the entire visibility graph but only part of it, therefore making the search process quicker. Before the V*GRAPH algorithm is presented, the A* algorithm will be briefly described in order to make the exposition of the V*GRAPH algorithm simpler and the proposition of the V*MECHA more thoroughly understood, more details about the A* algorithm are discussed in the Appendix B.

4.3.1 The A* Algorithm

The A* algorithm belongs to the family of *heuristic*² search engines for the establishment of the shortest path between two vertices in a graph and it was proposed

² The word *heuristic* comes from the Greek word *heuriskein*, meaning “to discover”.

by Hart *et al* (1968). Heuristics are criteria or principles for taking a decision upon several different actions and establish which of them is the most promising in order to achieve a goal more effectively. Consider the problem of finding the shortest path between vertex s and vertex g in an undirected graph with a total of n vertices. In order to give a physical hypostasis to the problem suppose that the vertices correspond to cities connected with roads (the edges) and that the shortest path between two cities is questioned.

The graph is not given in an explicit specification, such as an array (or list) of vertices and arcs (with the associated costs) but in an implicit specification, which is defined by a source vertex (the start vertex) s and a successor operator Γ . When the successor operator is applied to a vertex $\{n_i\}$ it produces a set $\{(n_j, c_{ij})\}$, where n_j are all the successors of n_i and c_{ij} is the cost of the associated edge from n_i to every n_j . It is then said that the vertices n_j are accessible from s . The process of applying the operator Γ to a vertex is called *expanding* a vertex.

The A* algorithm, searches for the minimum cost path (in this case the shortest path) by iteratively expanding vertices starting from vertex s . Each time a vertex n_i is expanded, an edge to each of its successors is created. In this manner a search graph G is explicitly generated, which is a part of an implicitly defined graph. For every successor n_j of n_i the minimum cost of getting to it, is calculated and if this successor has not been visited before (produced previously as successive vertex after the expansion of a different vertex) a pointer to its parent vertex is held, generating a search tree ST , which is a subset of G . Note that the search tree ST is a spanning tree of G . If this node has

been visited before then if the new way of attaining such a successor is less costly than the previous generated path to it, the algorithm redirects the pointers of this vertex towards n_i in the tree, updates the cost of the path to get there and reconsiders it for the establishment of the minimum cost path from s to g . The algorithm terminates when the goal vertex is reached or there are no more vertices for expansion. The minimum cost path from s to g (providing that there is one) is obtained by backtracking all the pointers from g to s in the search tree ST .

If an algorithm is guaranteed to return the optimal path (the shortest in this case) between two vertices in a δ graph³ (if there is one), otherwise it returns failure, it is called *admissible*. An evaluation function $\hat{f}(n)$ is defined for every vertex in the graph in such a way that it determines which vertex will be expanded next. This is an efficient way to ensure that vertices, which are not on the shortest path, are not expanded, and vertices that should be expanded are not ignored. A good choice for \hat{f} can guarantee A*'s admissibility.

Suppose that the function $f(n)$ defines the actual cost of the shortest path from s to g constrained to pass through n . If function $g(n)$ defines the actual cost of the shortest path from s to n and function $h(n)$ defines the actual cost of the shortest path from n to g then $f(n) = g(n) + h(n)$. The function f is not known in advance therefore an estimation function \hat{f} is used. To construct the estimation function \hat{f} , the estimation functions of g and h should be found and summed. A good estimation for $g(n)$ is where $\hat{g}(n)$ is the

³ δ graph is called a graph G when $\exists \delta > 0$ such as $\forall e_{ij} \in G: c_{ij} \geq \delta$, where e_{ij} is the edge connects the i vertex to the j vertex in the graph and c_{ij} is the cost associated with this edge.

minimum cost path from s to n the algorithm has found so far, therefore $\hat{g}(n) \geq g(n)$.

An estimation of the function $h(n)$ is not easy to find because the graph has not been explored yet beyond vertex n , so the best way to define it is to rely on heuristic information about the problem domain. For the shortest path between two cities a good estimation of $h(n)$ is for $\hat{h}(n)$ to be equal to the airline distance from n to g . Note that this distance is the smallest possible. The A* algorithm presented by Hart *et al* (1968) is as follows:

A* Algorithm

1. Mark s “open” and calculate $\hat{f}(s)$.
2. Select the open vertex n whose value of \hat{f} is the smallest. Resolve ties arbitrarily, but always in favour of vertex g .
3. If $n = g$, mark n “closed” and terminate the algorithm
4. Otherwise mark n closed and apply the successor Γ to n . Calculate \hat{f} for each successor of n and mark as open each successor not already marked closed. Remark as open any closed vertex n_i which is successor of n and for which $\hat{f}(n_i)$ is smaller now than it was when n_i was marked closed. Go to step 2.

When \hat{h} underestimates h then the A* algorithm is admissible. For a proof of the admissibility and optimality of the A* algorithm see Appendix B or (Hart *et al*, 1968). For more information about heuristic search strategies see (Nilsson, 1980), (Barr and Feigenbaum, 1983), (Pearl, 1984) and (Nilsson, 1998). In (Pearl, 1983), an attempt to quantify the knowledge-search trade off in heuristic search engines and specifically for the A* algorithm was made. An analysis of the average number of vertices expanded, as a function of the accuracy of its heuristic estimates, is provided. Having discussed the A* algorithm, the V*GRAPH algorithm can now be presented.

4.3.2 The V*GRAPH Algorithm

The principle idea behind the V*GRAPH algorithm is to keep the size of the visibility graph⁴ (i.e. the number of edges and vertices of the graph) as small as possible while it provides adequate representation of the free space (to be precise, of the semi-free space) for the establishment of the shortest semi-path between the AGV's start and goal locations. This can be achieved by expanding as few vertices as possible. In the V*GRAPH algorithm the authors also reduced the size of the visibility graph by observing that the shortest semi-free path never visits obstacles' vertices with obtuse⁵ interior polygon angle and that only visits vertices, which are extreme vertices of visible sequences. In order to give some insight to these definitions consider the environment of Figure 4.1.

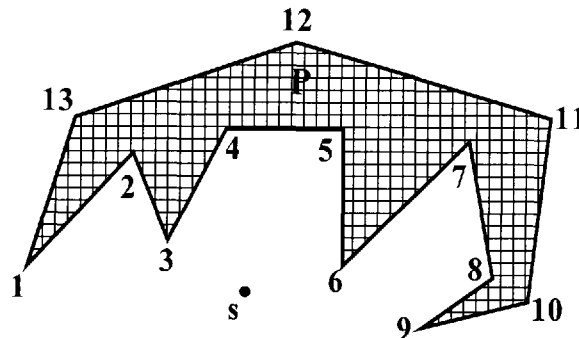


Figure 4.1 2, 4, 5, 7 and 8 are obtuse s-visible vertices. 1, 3, 6, 8 and 9 are the extreme vertices of the s-visible sequences.

⁴ Note that in the V*GRAPH algorithm as well as in V*MECHA (see section 4.6), the visibility graph is defined implicitly by the source vertex s and a successor operator Γ on s .

⁵ An obtuse angle is an angle which is greater than 90° and less than 180° . However, the authors in (Alexopoulos and Griffin, 1992) used the term obtuse angle to express an angle greater than 90° . This term is also adopted in this section to demonstrate the V*GRAPH algorithm.

All vertices that are visible from vertex s are called s -visible vertices. Obtuse s -visible vertices in Figure 4.1 are the following vertices: 2, 4, 5, 7 and 8. An s -visible sequence is a set of consecutive s -visible vertices on a single obstacle's boundary. s -visible sequences in Figure 4.1 are the following: $\{1\}$, $\{3, 4, 5, 6\}$, $\{8, 9\}$. Extreme vertices of the s -visible sequences are the following: 1, 3, 6, 8, and 9.

The V*GRAPH algorithm starts by expanding the AGV's start point s , it identifies all the s -visible vertices and then it places them on a list VV. It then rejects all the non-extreme vertices of any s -visible sequence and all the obtuse vertices from the list VV and places the remaining vertices on a list called OPEN. The list OPEN contains at any time all the vertices that are candidates for expansion next. The algorithm uses an evaluation function \hat{f} (similar to the one used by the A* algorithm) over each vertex on the list OPEN in order to choose for expansion the vertex which is most promising to lead to the goal point. The algorithm carries on in this manner iteratively until the goal point is reached (this is when it is picked from the list OPEN for expansion) or until there are no further candidate vertices for expansion. More formally the V*GRAPH algorithm for solving the basic movers' problem was stated as follows:

V*GRAPH Algorithm

INPUT:	Initial robot position, s , goal position, t and obstacles.
OUTPUT:	Shortest collision-free path P from s to t

```

begin
    P      := {s};
    OPEN := {s};
     $\hat{g}(s)$  := 0;
     $\hat{f}(s)$  := 0;
    repeat
        w := {i ∈ OPEN :  $\hat{f}(i) \leq \hat{f}(j), \forall j \in \text{OPEN}$ };
        remove w from OPEN;
        put w in P;
        VV := {w-visible vertices not on OPEN};
        EV := {extreme vertices of the w-visible paths in VV};
        AV := EV - {obtuse vertices in EV};
        for each vertex i in AV do
             $\hat{h}(i)$  := Euclidean distance d(i, g) from i to t;
             $\hat{g}(i)$  :=  $\hat{g}(w) + d(w, i)$ ;
             $\hat{f}(i)$  :=  $\hat{g}(i) + \hat{h}(i)$ ;
            put i on OPEN;
        until ((OPEN = 0) or (w = t));
        if (OPEN = 0) then exit with failure;
        if (w = t) then exit with P;
end.

```

The authors claimed that the V*GRAPH algorithm takes advantage of their observations in order to reduce the size of the visibility graph and that it also makes use of the A* algorithm in order to find the shortest semi-path between the robot's start and goal points in $O(n^2 \log n)$ computational time and $O(n^2)$ space, where n is the total number of the obstacles vertices. However, as it will be shown in section 4.5 one of the

observations for reducing the size of the visibility graph is incorrect. To be specific this observation is the one, which states that the shortest semi-path does not go via obtuse vertices. In addition, Conn *et al* (1997) constructed a counterexample showing that the V*GRAPH algorithm is incorrect due to the fact that it employs the A* algorithm incorrectly. In section 4.4 this counterexample will be demonstrated and it will also be shown that the V*GRAPH algorithm employs the A* algorithm incorrectly.

4.4 Conn *et al* (1997)'s Counterexample on the V*GRAPH Algorithm

The scenario of the counterexample given by Conn *et al* (1997), is as follows. Consider the planar environment of Figure 4.2, populated by a stationary simple obstacle P_1 , the robot's start point s and its goal point t . The co-ordinates of the obstacle's vertices, the robot's start and goal points are given in Table 4.1.

Points	Co-ordinates
s	(4, 2)
t	(4, 5)
1	(1, 2)
2	(5, 3)
3	(6.5, 1)
4	(6.5, 4)
5	(1, 4)

Table 4.1 The co-ordinates of the obstacle's vertices, the robot's start point s and its goal point t .

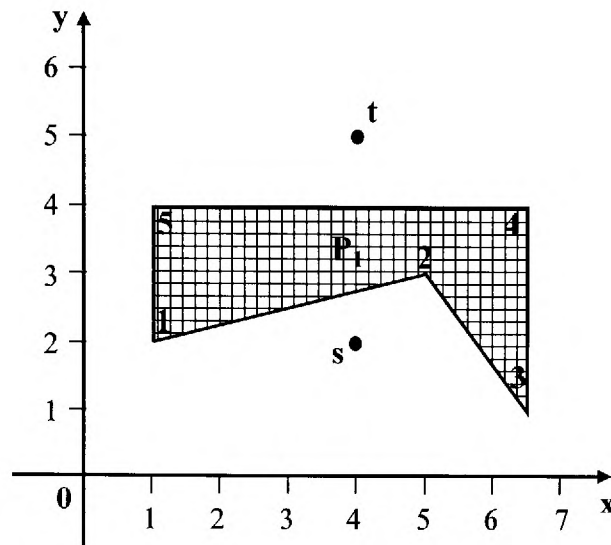


Figure 4.2 Illustration of the scene of the counter example.

To test the V*GRAPH algorithm the above example is fed to it as an input and the values of its variables at each iteration along with its outputs are presented in Table 4.2.

Iterations	i	P	OPEN	w	VV	EV	AV	$\hat{g}(i)$	$\hat{h}(i)$	$\hat{f}(i)$
		s	s							
1		s	\emptyset	s	1,2,3	1,3	1,3			
	1		1					3	4.242	7.242
	3		1,3					2.692	4.716	7.408
2		s,1	3	1	2,5	2,5	5			
	5		3,5					5	3.162	8.162
3		s,1,3	5	3	1,2,4	1,2,4	1,4			
	1		1,5					8.282	4.242	12.524
	4		1,4,5					5.692	2.692	8.384
4		s,1,3,5	1,4	5	t	t	t			
	t		1,4,t					8.162	0	8.162
5		s,1,3,5,t	1,4	t	5	5	5			
	5		1,4,5					11.324	3.162	14.486
		s,1,3,5,t		t						

Table 4.2 Summary of the results of the V*GRAPH algorithm when is applied to the scenario of Figure 4.2. The arrow indicates at which point the algorithm fails.

As can be noticed from Table 4.2, when the algorithm is applied to the scenario of Figure 4.2, it terminates with a path $P = \{s, 1, 3, 5, t\}$ of total length 8.162 units. It is obvious that there is a bug in the V*GRAPH algorithm because the shortest path it produced for a relatively easy environment (informally a relatively easy environment is populated by small number of obstacles which do not create narrow passages for the robot) is not collision-semi-free. Therefore, if the AGV moves along the produced path it will eventually collide with the environment's obstacle P_1 . Figure 4.3 illustrates the path produced by the V*GRAPH algorithm for the environment of Figure 4.2.

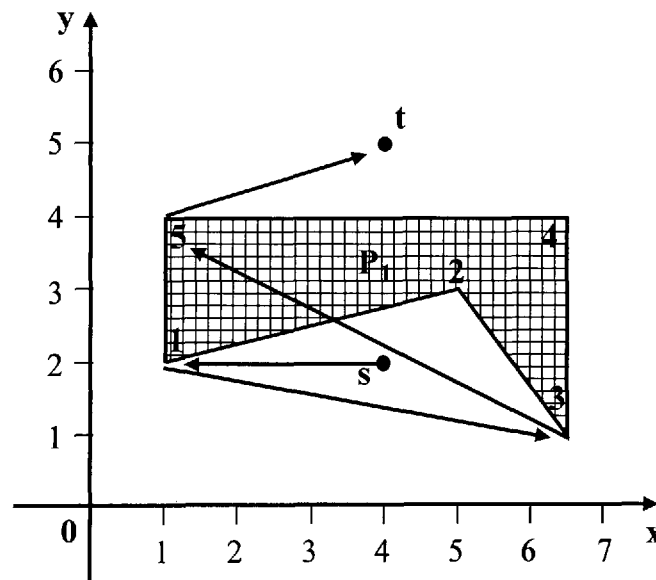


Figure 4.3 Illustration of the collision path produced by the V*GRAPH algorithm for the scenario of Figure 4.2.

The fact that a valid solution for the scenario of Figure 4.2 exists (this is the shortest semi-path between s and t) but the algorithm fails to produce it, implies that the algorithm is not complete and therefore incorrect. Note that one counterexample is

enough evidence to prove an algorithms' incompleteness. In section 4.4.1 the weak points of the algorithm will be identified and the reason for the algorithm's failure will be discussed.

4.4.1 The Bug in the V*GRAPH Algorithm

Recall from section 4.3.1 that in the A* algorithm for each visited vertex a pointer to its parent vertex in the search tree is maintained. Every time a vertex say n_{exp} is expanded it produces a vertex or a set of vertices, which are neighbouring to it. Suppose that after the expansion of the vertex n_{exp} , a vertex n_{neighb} is produced, if this vertex has not been visited before it is placed on OPEN for consideration for expansion next. If n_{neighb} has been visited before then, if the new way of attaining n_{neighb} is less time consuming than the previous one, the path is updated by redirecting n_{neighb} 's pointer towards n_{exp} , its path cost is updated and is placed on OPEN (if not already on OPEN) for reconsideration for expansion next. In this way at each iteration of the algorithm the produced path reflects the best path in the environment explored so far. These very important steps of the A* algorithm are not contained in the V*GRAPH algorithm, therefore any vertex, which is expanded by the V*GRAPH algorithm becomes a member of the final path resulting the algorithm producing non-optimal paths and more importantly non-collision-free paths.

This is exactly why the algorithm failed to produce a collision-free path in Conn *et al* (1996)'s counterexample, presented in section 4.4. Every vertex expanded by the V*GRAPH algorithm became member of the final path without maintaining a pointer to its parent node. Notice in table 4.2 that after the expansion of vertex 3, the algorithm chose for expansion vertex 5 and it placed it on the final path. As it can be seen in

Figure 4.3 the path between vertex 3 and vertex 5 is not collision-free and therefore the path produced by the algorithm ($P = \{s, 1, 3, 5, t\}$) is not collision-free. The algorithm failed at its fourth iteration when it expanded and placed vertex 5 in the final path immediately after vertex 3.

This enough evidence to conclude that the V*GRAPH algorithm does not employ the A* algorithm correctly, therefore it does not produce semi-free paths nor optimal paths as was claimed by its authors.

4.5 Identification of a New Deficiency on the V*GRAPH Algorithm

In (Alexopoulos and Griffin, 1992) a theorem was proposed and proved stating that: A shortest semi-free path from s to g cannot contain obtuse obstacles' vertices (according to their definition, vertices with interior polygonal angle greater than 90°).

This statement is not always true, because if an obstacle's vertex has interior polygon angle ϑ , with $\frac{\pi}{2} < \vartheta < \pi$, then this vertex is obtuse vertex, but at the same time it is non-concave, Figure 4.4c depicts such a vertex. As, it will be shown by Proposition 4.2 in section 4.6 the shortest path from s to g cannot contain concave vertices (vertices with interior polygon angle greater than π radians, Figure 4.4b), but this does not imply that it cannot contain obtuse non-concave vertices.

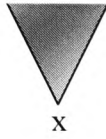


Figure 4.4a Vertex x is an acute vertex.

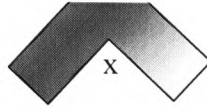


Figure 4.4b Vertex x is an obtuse concave vertex.

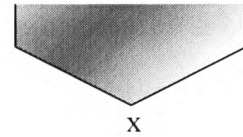


Figure 4.4c Vertex x is an obtuse non-concave vertex.

Therefore since the V*GRAPH algorithm rejects obtuse non-concave vertices it might mistakenly miss a path from s to g , which goes through an obtuse non-concave vertex. Figure 4.5 illustrates such a situation.

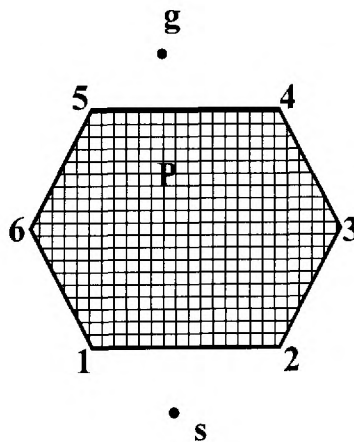


Figure 4.5 Illustration of an environment where the V*GRAPH algorithm fails to find a path, due to the fact that it does not consider obstacles' obtuse vertices for the construction of the visibility graph.

In section 4.6 an algorithm called V*MECHA is proposed for path planning of an AGV. The V*MECHA algorithm takes advantage of some useful observations for the reduction of the vertices considered for the construction of the visibility graph from the V*GRAPH algorithm, and as will be shown makes proper use of the A* algorithm to provide a solution to the basic movers' problem. It will be shown that the V*MECHA algorithm overcomes the deficiencies of V*GRAPH, and always returns the shortest semi-free path between two query points for an AGV providing that one exists.

4.6 Proposition of the V*MECHA Algorithm for Path Planning

As it was mentioned in section 4.1 the V*MECHA algorithm is proposed to overcome the deficiencies of the V*GRAPH algorithm. The V*MECHA algorithm is a completion of the V*GRAPH algorithm, it combines both the V*GRAPH and the A* algorithms to solve the basic movers' problem. The proposed algorithm solves Problem 1' as addressed in the introduction of the chapter (section 4.1) and makes use of Proposition 4.1 and Proposition 4.2 (see below) in order to construct a reduced visibility graph of the AGV's configuration space, thereby making the search process for a path between the AGV's start and goal configurations more time efficient. Before the presentation of the two propositions Theorem 4.1, which guarantees the optimality of the path, is discussed.

Theorem 4.1

Given the AGV's start point s , its goal point g and a set of static simple polygonal obstacles P , the shortest semi-free path ϕ between s and g (providing that s and g are in the same connected component of the AGV's free space) such that, $\forall P_i \in P, \phi \cap P_i = \emptyset$ is composed of straight line-segments joining s and g via the obstacles' vertices.

Proof

The proof is by contradiction. Let V be the set of the obstacles' vertices as well as s and g . Let E be the set of edges connecting vertices within V such that every edge in E does not intersect the interior of any obstacle. Suppose now, that the shortest semi-free path ϕ between s and g contains an edge, which does not join two vertices from V . Therefore at least one point p_i on the path ϕ exists, such that the path bends at p_i and $p_i \notin V$. Let p_{i-1} and p_{i+1} be the predecessor and the successor of p_i on the path respectively. Figure 4.6 illustrates such a path.

The polygonal line p_{i-1}, p_i and p_{i+1} is a convex sub-segment of the polygonal line s, p_{i-1}, p_i, p_{i+1} and g . Since ϕ is the global shortest path, according to the principle of optimality the sub-path ξ of ϕ (this is the path from p_{i-1} to p_{i+1} through p_i) is the shortest between p_{i-1} and p_{i+1} . However, note that if the straight line-segment $\overline{p_{i-1}p_{i+1}}$ is collision-free (to be more precise semi-free), then $\overline{p_{i-1}p_{i+1}}$ is shorter than any other polygonal line with the same end points and therefore is shorter than the sub-path ξ , hence ϕ is not optimal and the theorem holds. If the straight line-segment $\overline{p_{i-1}p_{i+1}}$ is not collision-free then a non-empty set $V' \subset V$ exists such that $\forall v_i \in V', v_i$ is inside the

triangle $\Delta p_{i-1} p_i p_{i+1}$. The convex polygonal subset p_{i-1}, v_1, \dots, v_k and p_{i+1} of the convex hull of the set of points $\{V' \cup p_{i-1} \cup p_{i+1}\}$ is shorter than any other polygonal line-segment with the same end points, which does not intersect the interior of the convex hull. Therefore, since the polygonal line p_i, p_{i-1} and p_{i+1} lies outside the convex hull of $\{V' \cup p_{i-1} \cup p_{i+1}\}$ the polygonal line p_{i-1}, v_1, \dots, v_k and p_{i+1} is shorter than the polygonal line p_{i-1}, p_i and p_{i+1} (convex hull definition) therefore, the sub-path ξ is not the shortest hence ϕ is not optimal and thus the theorem holds.

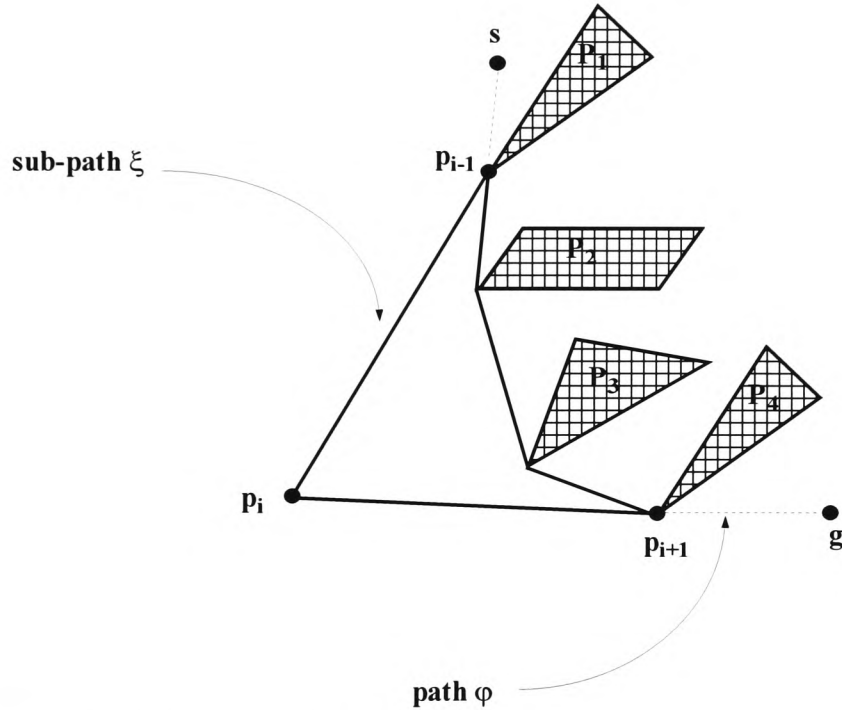


Figure 4.6 The shortest path bends only at the obstacles' vertices.

Theorem 4.1 entails that the shortest semi-free path between two given points in an environment populated by polygonal obstacles is contained in the environment's visibility graph. Therefore the shortest path in the visibility graph between vertices s

and g is the actual shortest path in the environment between the AGV's start and goal points.

Recall from section 4.3.2 that the authors of the V*GRAPH algorithm reduced the size of the visibility graph by only considering the extreme vertices of any visible sequence for its construction. A similar argument is used by the V*MECHA algorithm for the reduction of the visibility graph. This argument will be proved in Lemma 4.1 and will be further extended by Proposition 4.1 to reduce even more the size of the visibility graph. Before the presentation and proof of Lemma 4.1 some useful definitions should be provided. Note that the proof of Lemma 4.1 is similar to the proof in (Alexopoulos and Griffin, 1992).

Recall from section 4.3.2 that all vertices that are visible from vertex s are called s -visible vertices. An s -visible sequence is a set of consecutive s -visible vertices on a single obstacle's boundary. The s -visible sequences in Figure 4.7 are the following: $\{1, 2, 3\}$, $\{15, 16\}$ and $\{9, 10\}$. Extreme vertices of the s -visible sequences are the following: 1, 3, 15, 16, 9, and 10. Notice that, since there can be more than one s -visible sequence for one obstacle there can be more than two extremes for a single obstacle. For instance, in Figure 4.7 there are two s -visible sequences for obstacle P_1 and these are $\{1, 2, 3\}$ and $\{9, 10\}$. The extreme vertices of the s -visible sequences of obstacle P_1 are 1, 3, 9 and 10.

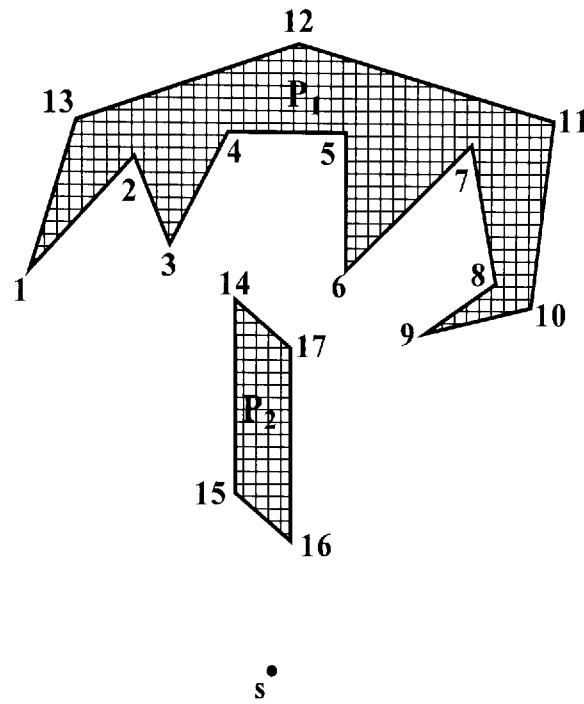


Figure 4.7 1, 3, 9, 10, 15 and 16 are the extreme vertices of the s-visible sequences.

The two extremes of all the extreme vertices of the s-visible sequences for a single obstacle are called super-extremes. In Figure 4.7 the super-extremes are the vertices 1, 10, 15, 16. Vertices 1 and 10 are the super-extremes for obstacle P_1 and the vertices 15 and 16 are the super-extremes of obstacle P_2 . Note that from a vertex s there are at most two super-extremes for each obstacle. In Lemma 4.1 will be proved that the shortest path between two query points for an AGV, in an environment populated by simple polygonal obstacles, goes via the extreme vertices of the visible sequences. Further in Proposition 4.1 it will be shown that the shortest path only visits the super-extremes of the extreme vertices of the visible sequences of every obstacle.

Lemma 4.1

From a vertex κ , only the extreme vertices of the κ -visible sequences need to be considered for the construction of the visibility graph because the shortest semi-free path from s to g through κ passes via the extremes vertices of any κ -visible sequence.

Proof

Consider the environment of Figure 4.8 and the κ -visible sequence on the obstacle P_1 .

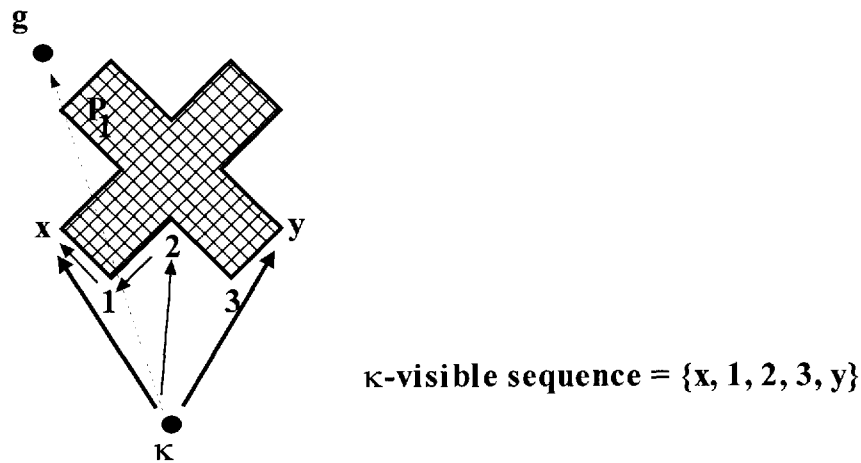


Figure 4.8 The shortest path between s and g though κ only visits extreme vertices of the κ -visible sequences.

The κ -visible sequence contains five vertices with extremes at its two ends, vertices x and y respectively. Note that if the sequence had less than three vertices all of them would be extremes. The shortest path between κ and the goal point g is the straight line-segment, which connects them. Unfortunately this path is not collision-free because it intersects the interior of the P_1 , therefore the AGV would have to circumnavigate P_1 . Since the AGV will go through point κ and will circumnavigate P_1

it will definitely pass either through vertex x or through vertex y . Therefore the shortest semi-path to the goal, which goes through point κ will contain either the line-segment $\overline{\kappa x}$ or the line-segment $\overline{\kappa y}$, because the straight line-segment between two points is shorter than any other polygonal line, which connects these points.

Indeed, as can be noticed from Figure 4.8 if the shortest semi-path to the goal through point κ , is the one which goes around the left hand side of P_1 , it will definitely go through vertex x and the straight line-segment $\overline{\kappa x}$ is shorter than any other polygonal line from κ to x through non-extreme vertices of the κ -visible sequence. Therefore the Lemma holds.

Proposition 4.1 is proposed to strengthen Lemma 4.1 and to further reduce the number of vertices considered for the construction of the visibility graph. Using Proposition 4.1 the size of the visibility graph produced by the V*MECHA algorithm is smaller than the visibility graph produced by the V*GRAPH algorithm.

Proposition 4.1

From a vertex κ , only the super-extremes of the extreme vertices of the κ -visible sequences for each obstacle need to be considered for the construction of the visibility graph because the shortest semi-free path from s to g through κ passes via the super-extremes of the κ -visible sequences.

Proof

The proof of the Proposition is very similar to that of Lemma 4.1. Consider the environment of Figure 4.9 and the κ -visible sequences on the obstacle P_1 . There are three κ -visible sequences and these are $\{1\}$, $\{3, 4, 5, 6\}$ and $\{8, 9\}$.

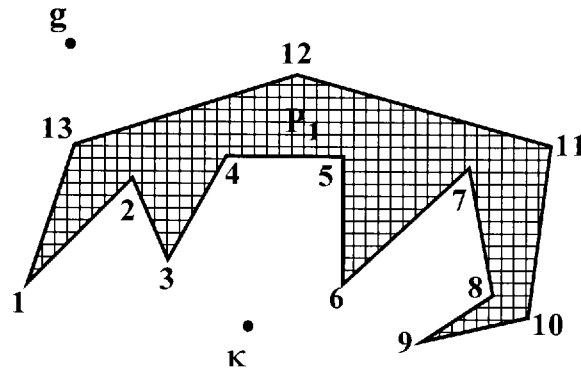


Figure 4.9 The super-extremes of the κ -visible sequences are the vertices 1 and 9.

According to Lemma 4.1 only the extreme vertices of the κ -visible sequences should be considered for the construction of the visibility graph. Extreme vertices of the κ -visible sequences are 1, 3, 6, 8 and 9.

It is known that the shortest path between κ and the goal point g is the straight line-segment, which connects them. Unfortunately this path is not collision free because it intersects the interior of the P_1 , therefore the AGV would have to circumnavigate P_1 . Notice that since the AGV will go through point κ and will circumnavigate P_1 it will definitely pass either through vertex 1 or through vertex 9, which are the super-extremes of the extreme vertices of the κ -visible sequences for P_1 . Therefore using the same

arguments as in Lemma 4.1 it easily follows that only the super-extremes of the extreme vertices of the visible sequences for each obstacle should be considered for the construction of the visibility graph.

Note that when the algorithm identifies the super-extremes of the k -visible sequences, it then marks all the non-super-extremes as useless, so they are not considered again later by the algorithm, if they are visible from a different vertex, say r , regardless whether they are super-extremes or non-super-extremes of the r -visible sequences. The reason for rejecting permanently the non-super-extreme vertices of a visible sequence is as follows.

Suppose that vertex, say v , is a non-super-extreme for an obstacle of the κ -visible sequences. If v is visible from another vertex later in the algorithm, say vertex r accessible from κ then if v is non-super-extreme vertex of the r -visible sequences it will be rejected anyway. If v is a super-extreme vertex of the r -visible sequences then it does not need to be considered for the construction of the visibility graph because, the line-segment connecting κ to v is shorter than any other path from κ to v through r . But since v is a non-super-extreme vertex in the κ -visible sequences this means that it does not need to be considered for the construction of the visibility graph. Therefore once non-super-extremes of the visible sequences are identified as the algorithm proceeds, they can be rejected permanently.

Proposition 4.2

The shortest semi-free path from the AGV's start point s to its goal point g , never visits concave obstacles' vertices.

Proof

It is trivial to show by contradiction that any path through a concave vertex is not the shortest. Consider the environment of Figure 4.10 and suppose the contrary, namely that the shortest semi-free path goes through the concave vertex x of the obstacle P_1 .

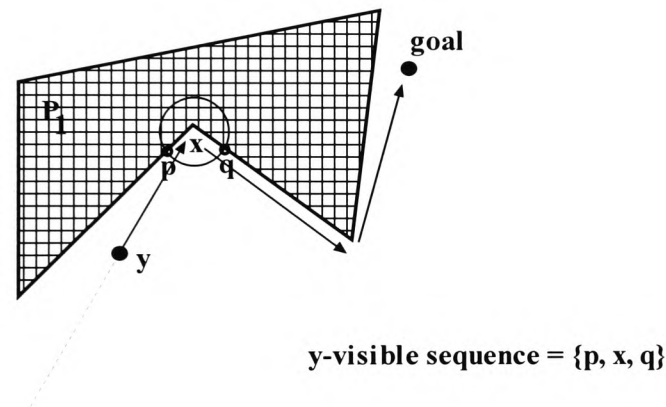


Figure 4.10 The shortest path never visits concave vertices.

Further suppose that the predecessor of x on the path is vertex y . Since obstacle P_1 is a simple polygon, a small real ε always exists, such that if a circle with radius ε and centre the vertex x is inscribed, it intersects the two adjacent edges of vertex x at points say p and q , which are visible from y . Therefore all three points p, x, q are consecutive points on P_1 's boundary and are visible from y . So these points constitute a y -visible sequence with extremes p and q . According to Lemma 4.1 the shortest semi-free path only visits the extreme vertices of a visible sequence, therefore the path through vertex

x can be shortened by passing either through point p or through point q instead and therefore is not the shortest, which contradicts the initial assumption that the path through the concave vertex x is the shortest and proves the Proposition.

Proposition 4.2 not only reduces the number of vertices considered for the construction of the visibility graph but also overcomes the deficiency that appears in the V*GRAPH algorithm which rejects the obtuse non-concave vertices.

From Proposition 4.1 and Proposition 4.2 it is concluded that all the concave vertices and all the non-super-extreme vertices of the visible sequences in the environment should be rejected and not considered by the algorithm, because the shortest semi-free path between two query points does not go via such vertices. Note that special treatment is needed when the AGV's goal point g coincides with a concave vertex or with a non-super-extreme vertex of an obstacle (i.e. for the g vertex not to be rejected).

To demonstrate the effectiveness of Proposition 4.1 and Proposition 4.2, consider the two-dimensional environment of Figure 4.11 s and g denote the AGV's start and goal locations respectively and P_1 and P_2 are the environment's obstacles.

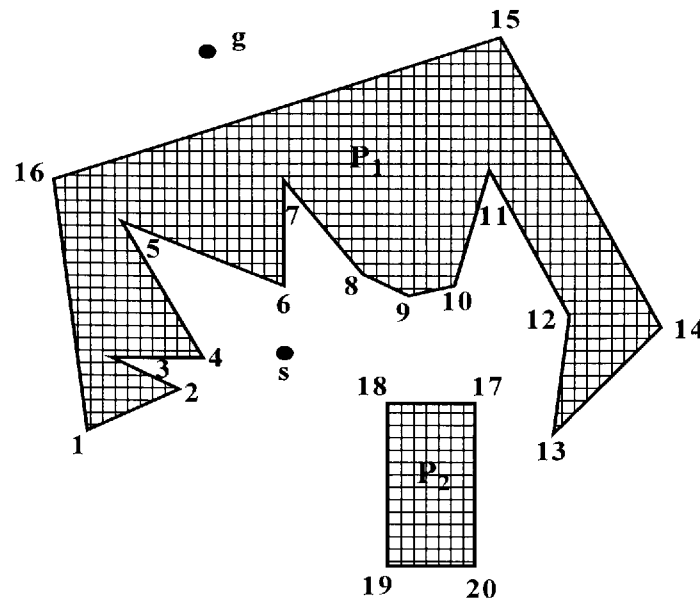


Figure 4.11 Illustration of the reduction due to Proposition 4.1 and Proposition 4.2.

Visible vertices from s are the vertices, 1, 2, 4, 5, 6, 7, 8, 9, 10, 12, 17, 18 and 19. There are four s -visible sequences, these are, $\{1, 2\}$, $\{4, 5, 6, 7, 8, 9, 10\}$, $\{12\}$ and $\{17, 18, 19\}$. By applying Proposition 4.1 to the environment of Figure 4.11 the only s -visible vertices, which need to be included in the visibility graph are the super-extremes, 1, 12, 17 and 19. From these vertices, vertex 12 is a concave vertex. So by applying Proposition 4.2 to the environment of Figure 4.11, the only s -visible vertices that are left to be included in the visibility graph are the vertices, 1, 17, and 19. In Figure 4.11, can be noticed the large reduction of vertices considered for the construction of the visibility graph, from 1, 2, 4, 5, 6, 7, 8, 9, 10, 12, 17, 18 and 19 to 1, 17 and 19. Having set and proven the two propositions responsible for the reduction of the vertices considered for the construction of the visibility graph, the proposed algorithm can now be presented.

4.6.1 The V*MECHA Algorithm

As was mentioned earlier in this section the V*MECHA algorithm makes use of the A* algorithm for the construction of the shortest semi-free path. The algorithm starts from the AGV's start configuration and iteratively generates a reduced visibility graph, which is searched for the identification of the shortest Euclidean path between s and g . The algorithm starts by expanding point s , it identifies all the s -visible vertices and then it rejects all the non-super-extremes of the s -visible sequences and all the concave vertices. The algorithm undertakes a heuristic search process guided by the evaluation function \hat{f} . Inspired by the A* algorithm, the evaluation function is defined as follows.

The evaluation function \hat{f} is defined such that its value $\hat{f}(n)$ for any vertex n is an estimation of $f(n)$. The function $f(n)$ is the cost of the actual shortest path from s to g constrained to pass through vertex n . If function $g(n)$ defines the actual cost of the shortest path from s to n and function $h(n)$ defines the actual cost of the shortest path from n to g then the function $f(n)$ is defined as follows,

$$f(n) = g(n) + h(n) \quad (4.1)$$

However, since is not possible to know $f(n)$ in advance, an estimation function is used instead. The estimation function of f is the function \hat{f} and in order to be defined the estimation functions of g and h should be defined and summed together. As in the A* algorithm described in section 4.3.1, a good estimation for $g(n)$ is $\hat{g}(n)$, where $\hat{g}(n)$ is the shortest path from s to n the algorithm has found so far, therefore $\hat{g}(n) \geq g(n)$. An estimation of the function $h(n)$ is not easy to find, so the best way to define it is to rely

on heuristic information about the problem domain. For the basic movers' problem a good estimation of $h(n)$ is for $\hat{h}(n)$ to be equal to the airline distance from n to g . Note that this distance is the smallest possible between two vertices in the environment. Therefore more formally the estimation function that is used can be defined as follows:

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n) \quad (4.2)$$

Having the evaluation function for the search process defined the algorithm proceeds as follows. When the algorithm expands s at the beginning, it rejects all the non-super-extremes of the s -visible sequences and all the concave vertices. All the (remaining) produced vertices are placed on a list called OPEN and a function \hat{f} is evaluated for each produced vertex in order for the vertex with the best assessment to be chosen for expansion next. The list OPEN contains at any time all the vertices that are candidates for expansion next. Every vertex produced is marked visited and is placed in a search tree called Path with a pointer to its parent vertex. The algorithm then picks from OPEN the vertex with the smallest \hat{f} value to expand next. The algorithm carries on in this manner iteratively until the goal point is reached (this is when it is picked from the OPEN list for expansion) or until there are no further candidate vertices for expansion on OPEN. Note that if after the expansion of the vertex say n_{exp} , a vertex n_{neighb} is produced, then if this vertex has been visited before and the new way of attaining it gives rise to a shorter path than that previously encountered, then the tree Path is updated by redirecting n_{neighb} 's pointer towards n_{exp} , its \hat{f} value is updated and placed on OPEN if its not already there, for reconsideration for expansion next. In this way at each iteration of the algorithm the tree Path reflects the best path in the environment

explored so far. If the algorithm terminates because the vertex g is reached, the shortest semi-free path between s and g is obtained by backtracking all the pointers in the tree Path from g to s . Otherwise if the algorithm terminates when the list OPEN is empty, this means that there are no further vertices for expansion and therefore there is not a semi-free path between s and g . The V*MECHA algorithm is proposed as follows:

V*MECHA Algorithm

INPUT: AGV's start point s , AGV's goal point g and the set of obstacles P .
OUTPUT: Shortest semi-free path for the AGV from s to g .

```

begin
    put  $s$  in Path;
    put  $s$  in Open;
    mark  $s$  visited;
     $\hat{g}(s) := 0$ ;
    while (Open  $\neq$  nil) do
        begin
             $w := \{i \in \text{Open} : \hat{f}(i) < \hat{f}(j) \mid \forall j \in \text{Open}, \text{ resolve ties arbitrarily but}$ 
always in favour of the goal vertex};
            remove  $w$  from Open;
            if  $w = g$  then exit while loop;
             $VV := \{ \text{VISIBLE\_VERTICES}(w, P) \}$ ;
             $SE := \{ \text{SUPER\_EXTREMES}(w, VV) \}$ ;
            Mark all the vertices in  $\{VV-SE\}$  useless;
             $NCV := SE - \{\text{concave vertices in } SE\}$ ;
            for each vertex  $i \in NCV$  do
                if  $i$  is not marked useless then
                    if  $i$  is not marked visited then
                        begin

```

```

 $\hat{h}(i)$  := Airline distance  $d_{air}(i, g)$  from  $i$  to  $g$ ;
 $\hat{g}(i)$  :=  $\hat{g}(w) + d_{edge}(w, i)$ ;
 $\hat{f}(i)$  :=  $\hat{g}(i) + \hat{h}(i)$ ;
put  $i$  in Path with pointer toward  $w$ ;
put  $i$  in Open;
mark  $i$  visited;

end;

else if  $\hat{g}(i) > \hat{g}(w) + d_{edge}(w, i)$  then
    begin
        redirect pointer of  $i$  toward  $w$  in Path;

        if  $i \in \text{Open}$  then remove  $i$  from Open;
         $\hat{g}(i)$  :=  $\hat{g}(w) + d_{edge}(w, i)$ ;
         $\hat{f}(i)$  :=  $\hat{g}(i) + \hat{h}(i)$ ;
        put  $i$  in Open;
    end;

end;

if  $w = g$  then return the path by tracing all the pointers in Path from  $g$  back to
 $s$  else if Open = nil then return failure;

end.

```

VV in the algorithm is a set, which contains all the visible vertices from the vertex currently being expanded. For the identification of the visible vertices from a vertex say v , an algorithm, which is due to Lee (1978) can be used. This algorithm is called as a subroutine by the V*MECHA algorithm called *VISIBLE_VERTICES* and will be discussed in section 4.6.2.

SE in the algorithm is a set, which contains the super-extremes of the extreme vertices of the visible sequence(s) in VV. For the identification of the super-extremes of the visible sequence(s), the algorithm uses the subroutine ***SUPER_EXTREMES***. This subroutine will be discussed in section 4.6.2.

NCV in the algorithm is a set, which contains only the non-concave vertices from SE.

$d_{\text{edge}}(w, i)$ is the cost (length) of the edge connecting vertices w and i .

$d_{\text{air}}(w, i)$ is the airline (Euclidean) distance between vertices w and i .

All vertices considered for the construction of the RVG are marked as visited.

All the non-super-extreme of the visible sequences are marked as useless

4.6.2 V*MECHA's Subroutines

The first subroutine used by V*MECHA is the ***VISIBLE_VERTICES***. This routine takes as arguments a point v and a set of obstacles P and returns a set W of all the vertices that are visible from v . The second subroutine used by the V*MECHA algorithm is the ***SUPER_EXTREMES***. This routine takes as arguments a point v and a set of its visible vertices W and returns all the super-extremes of the v -visible sequence(s).

VISIBLE VERTICES (v, P)

INPUT: A collision-free point v and the set of obstacles P .

OUTPUT: The set W of all vertices visible from vertex v .

begin

Sort the scene's vertices according to their clockwise angle the half-line emanating from v through each vertex creates with the x-axis. If there are any ties give priority according their distance to v . Let p_1, p_2, \dots, p_n be the sorted list;

Let ℓ be the half-line emanating from v and parallel to the x-axis. Find the obstacle's edges that are properly intersected by ℓ (intersected in other points than the obstacle's edges endpoints) and store them in a balanced tree T in the order their intersection;

$W := \emptyset$;

for $i := 1$ **to** n **do**

if **VISIBLE** (p_i) **then** add p_i to W ;

Insert into T the obstacle's edge incident to p_i that lie on the clockwise side of the half-line from v to p_i ;

Delete from T the obstacle's edge incident to p_i that lie on the counter-clockwise side of the half-line from v to p_i ;

return W ;

end.

The *VISIBLE* subroutine is called by the *VISIBLE_VERTICES* subroutine, and it is used to decide whether a vertex p_i is visible from v . Usually this involves only checking within the tree T if the edge closest to v (note that this is the leftmost leaf of the tree)

intersects $\overline{vp_i}$. However the visible subroutine should take care of the case when $\overline{vp_i}$ contains other vertices. The *VISIBLE* subroutine is as follows:

***VISIBLE* (p_i)**

```

begin
  if  $\overline{vp_i}$  intersects the interior of the obstacle of which  $p_i$  is a vertex, locally at  $p_i$ 
    then return false
  else if  $i=1$  or  $p_{i-1}$  is not on the segment  $\overline{vp_i}$ 
    then Search in T for the edge e in the leftmost leaf
      if e exists and  $\overline{vp_i}$  intersects e
        then return false
      else return true
    else if  $p_{i-1}$  is not visible
      then return false
    else Search in T for an edge e that intersects  $\overline{p_{i-1}p_i}$ 
      if e exists
        then return false
      else return true
end.

```

By analysing the computational time of the *VISIBLE_VERTICES* subroutine, it can be noticed that the first step of the algorithm, which sorts the vertices according to their clockwise angle requires $O(n \log n)$ computational time, where n is the total number of the obstacles' vertices. The second step, which checks for intersection between the obstacles edges and the half-line, which emanates from v and is parallel to the x-axis and places the corresponding edge in the a balanced search tree T , requires $O(n \log n)$ computational time, where n is the total number of the obstacles edges. The last step is

the execution of the **for** loop. In each execution of the loop constant time is required for some geometric checks (these checks are carried out in the subroutine *VISIBLE*) and $O(\log n)$ time for a constant number of operations on the balanced tree T , where n is the total number of the obstacles' vertices. Since there can be at most n iterations in the **for** loop, its computational time $O(n \log n)$, where n is the total number of the obstacles' vertices. Therefore the overall computational time of the *VISIBLE_VERTICES* subroutine is $O(n \log n)$, where n is the total number of vertices of the input set of obstacles P .

SUPER_EXTREMES Subroutine

Before the *SUPER_EXTREMES* subroutine is proposed it is important to note that this subroutine requires the vertices of the obstacles to be numbered consecutively around the obstacles (say in counter-clockwise order). Each vertex is numbered by a set of two digits. The first digit is the index of the obstacle and the second is the index of the vertex. For example, suppose that a scene contains two obstacles each of them having three vertices. The vertices of the first obstacle are numbered counter-clockwise as, (1, 1) for the first vertex, (1, 2) for the second and (1, 3) for the third. The vertices of the second obstacle are numbered counter-clockwise as, (2, 1), (2, 2) and (2, 3). Note that this way of representing the obstacles' vertices does not influence the V*MECHA algorithm or any of its subroutines in the way they are executed or their computational complexity. Therefore, it is assumed that the input of the problem is fed to the algorithm in the above manner. However, in the rest of the thesis except the following subroutine the vertices will continue to be numbered as they were up to this point (by a single digit) for the sake of the uniformity and the clarity of the exposition.

The ***SUPER_EXTREMES*** subroutine takes as arguments a point v and a set of its visible vertices W and returns all the super-extremes of the extreme vertices of the v -visible sequence(s). The routine compares the counter-clockwise angles created by the line-segments that connect v to any of the v -visible vertices, with the x -axis, in order to find the super-extremes. This is the reason that the obstacles' vertices should be numbered using two digits so that the routine can distinguish the vertices of different obstacles and produce the super-extremes of each obstacle.

Notice that the super-extremes can be identified by the ***SUPER_EXTREMES*** subroutine directly without having to identify the extreme vertices of the visible sequences at all.

SUPER_EXTREMES (v, W)

INPUT: A collision-free point v and a set W of v -visible vertices.
OUTPUT: A set with the super-extremes of the v -visible sequence(s) in W .

begin

Put all vertices from W to Temp sorted lexicographically upon their two digits numbering;

$k := 1$;

$left_k :=$ first element Temp;

$right_k :=$ first element Temp;

for $v_i :=$ (the second element of Temp) **to** (the last element of Temp) **do**

begin

if $v_i(\text{obs}) \neq v_{i-1}(\text{obs})$ **then** $k := k + 1$;

```

if  $\text{left}_k = \emptyset$  and  $\text{right}_k = \emptyset$  then  $\text{left}_k := v_i(\text{obs}, \text{ver})$  and  $\text{right}_k$ 
 $:= v_i(\text{obs}, \text{ver})$  else if the counter-clockwise angle created by  $\overline{vv_i}$  and
the x-axis is greater than this of the  $\text{left}_k$  then  $\text{left}_k := v_i(\text{obs}, \text{ver})$  else if
the counter-clockwise angle created by  $\overline{vv_i}$  and the x-axis is smaller
than this of the  $\text{right}_k$  then  $\text{right}_k := v_i(\text{obs}, \text{ver})$ ;

end;
end.

```

At the end of the above subroutine, k is the number of the super-extremes pairs and left_k and right_k are the left and right super-extremes respectively for each encountered obstacle. For each vertex v_i in W , obs and ver are its obstacle's number and vertex's number indexes. In order to demonstrate how the above routine works, a simple example will be given. Consider the example of Figure 4.12.

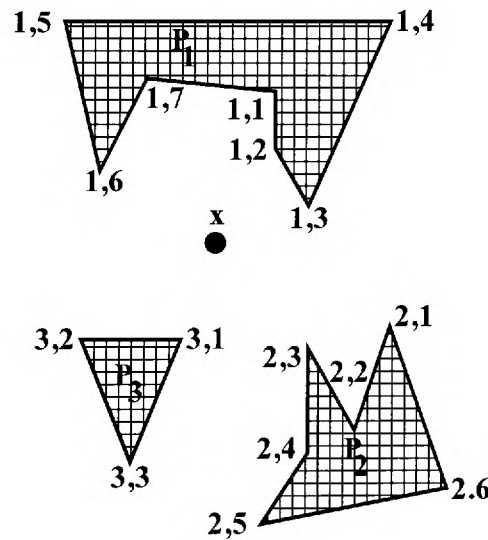


Figure 4.12 Environment used for the demonstration of the *EXTREME_VERTICES* subroutine.

In the example of Figure 4.12, $W = \{(2, 1), (2, 3), (2, 4), (2, 5), (3, 1), (3, 2), (1, 6), (1, 7), (1, 1), (1, 2), (1, 3)\}$ and $Temp = [(1, 1), (1, 2), (1, 3), (1, 6), (1, 7), (2, 1), (2, 3), (2, 4), (2, 5), (3, 1), (3, 2)]$ at the beginning of the algorithm. The results of the algorithm when applied to the example of Figure 4. are summarized in Table 4.3.

Iterations	v_i	k	$left_k$	$right_k$
		1	(1, 1)	(1, 1)
1	(1, 2)			(1, 2)
2	(1, 3)			(1, 3)
3	(1, 6)		(1, 6)	
4	(1, 7)			(1, 7)
5	(2, 1)	2	(2, 1)	(2, 1)
6	(2, 3)			(2, 3)
7	(2, 4)			(2, 4)
8	(2, 5)			(2, 5)
9	(3, 1)	3	(3, 1)	(3, 1)
10	(3, 2)			(3, 2)

$\left. \begin{array}{l} \text{Iteration 1} \\ \text{Iteration 2} \\ \text{Iteration 3} \\ \text{Iteration 4} \end{array} \right\} \text{left}_1 = (1, 6) \quad \text{right}_1 = (1, 3)$

$\left. \begin{array}{l} \text{Iteration 5} \\ \text{Iteration 6} \\ \text{Iteration 7} \\ \text{Iteration 8} \end{array} \right\} \text{left}_2 = (2, 1) \quad \text{right}_2 = (2, 5)$

$\left. \begin{array}{l} \text{Iteration 9} \\ \text{Iteration 10} \end{array} \right\} \text{Left}_3 = (3, 1) \quad \text{right}_3 = (3, 2)$

Table 4.3 Summarization of the results of the *SUPER_EXTREMES* subroutine when is applied is applied to the scene of Figure 4.12.

As can be notice from Table 4.3, when the routine *SUPER_EXTREMES* is applied to the example of Figure 4.12, it terminates with three super-extremes pairs ($k = 3$) and these are, $[(1, 6), (1, 3)]$, $[(2, 1), (2, 5)]$ and $[(3, 1), (3, 2)]$.

By analysing the computational time of the *EXTREME_VERTICES* subroutine it can be noticed that the first step of the algorithm, which sorts the vertices first by their obstacle's index obs and vertex's index ver and places them in Temp, requires $O(n \log n)$ computational time, where n is the total number of the obstacles' vertices. The second step, is the loop **for** which encapsulates statements of constant time and is

traversed n times. Therefore the overall computational time of the *EXTREME_VERTICES* subroutine is $O(n \log n)$, where n is the total number of vertices of the input set of obstacles P .

Having discussed the subroutines of the V*MECHA algorithm its presentation is completed and it is now appropriate to test the algorithm by considering a path planning problem. Before any formal proof of the algorithm's admissibility and optimality, the algorithm will be tested in section 4.7 on the counterexample of the V*GRAPH algorithm constructed by Conn *et al* (1997).

4.7 Testing V*MECHA using Conn *et al* (1997)'s Counterexample

A quick reminder of the counterexample's scene is provided in Figure 4.13 and a summary of the scene's co-ordinates is provided in Table 4.4.

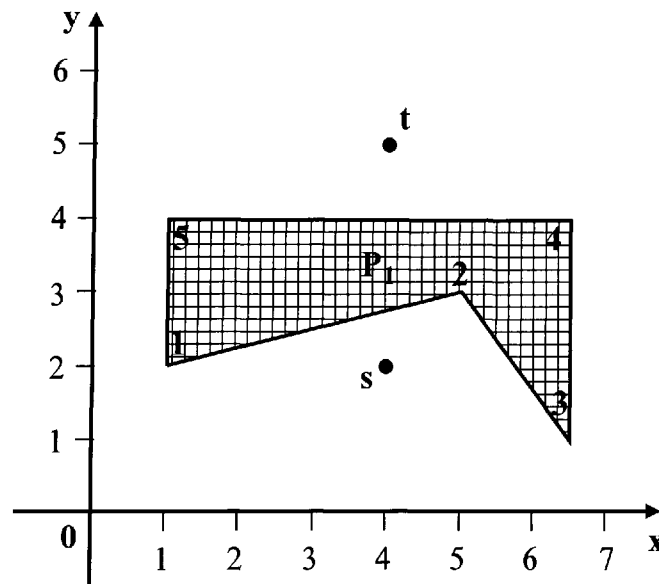


Figure 4.13 Illustration of the counterexample's scene.

Points	Co-ordinates
s	(4, 2)
t	(4, 5)
1	(1, 2)
2	(5, 3)
3	(6.5, 1)
4	(6.5, 4)
5	(1, 4)

Table 4.4 The co-ordinates of the scene's vertices.

To test V*MECHA the aforementioned example is fed into it as input and the values of its variables at each iteration along with its outputs are presented in Table 4.5.

At the first iteration of the algorithm vertex s is expanded and after the filtering of the s-visible vertices using proposition 4.1 and 4.2, vertices 1 and 3 are added to the spanning tree Path with a pointer pointing towards s (see table 4.5 for details). At the second iteration vertex 1 is chosen from Open to be expanded because it has the smallest \hat{f} value. After filtering the 1-visible vertices using proposition 4.1 and 4.2, and checking for the already visited vertices (these are vertices s and 3) whether the new way of getting to them is less time consuming than the previous, the only vertex which is added to the Path, is vertex 5 with a pointer pointing towards vertex 1. In the same manner at the third iteration vertex 3 is chosen from Open to be expanded and vertex 4 is added to the path with a pointer pointing towards vertex 3. At the fourth iteration vertex 5 is chosen to be expanded and vertex g is added to the path with a pointer pointing towards vertex 5. At the fifth iteration vertex g is chosen to be expanded next and therefore the algorithm terminates because the goal point is reached.

Iterations	i	OPEN	w	VV	SE	NCV	useless	Visited	$\hat{g}(i)$	$\hat{h}(i)$	$\hat{f}(i)$
1		s	s	1,2,3	1,3	1,3	2	s			
	1	\emptyset						s,1	3	4.242	7.242
	3	1,3						s,1,3	2.692	4.716	7.408
2		3	1	s,2,3,5	s,3,5	s,3,5	2				
	s								? 0 > 3+3		
	3								? 2.692 > 3+5.590		
	5	3,5							5	3.162	8.162
3		5	3	s,1,2,4	s,1,4	s,1,4	2				
	s								? 0 > 2.692+2.692		
	1								? 3 > 2.692+5.590		
	4	4,5						s,1,3,4,5	5.692	2.692	8.384
4		4	5	1,4,g	1,4,g	1,4,g					
	1								? 3 > 5+2		
	4								? 5.692 > 5+5.5		
	g							s,1,2,3,4,5,g	8.162	0	8.162
5		1									
	4,g										
	g										

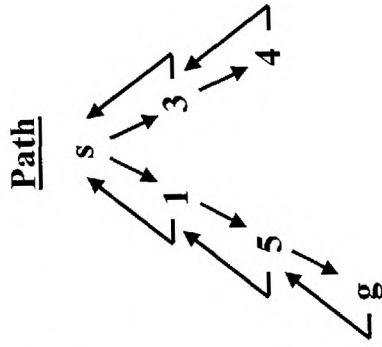


Table 4.5 Summary of the results of the V*MECHA algorithm when is applied to the scenario of Figure 4.13.

As can be noticed from Table 4.5, when the V*MECHA algorithm is applied to the scenario of Figure 4.13, it terminates with a path $\text{Path} = \{s, 1, 5, g\}$ of total length 8.162 units. Figure 4.14 depicts the path produced by the V*MECHA algorithm for the environment of Figure 4.13.

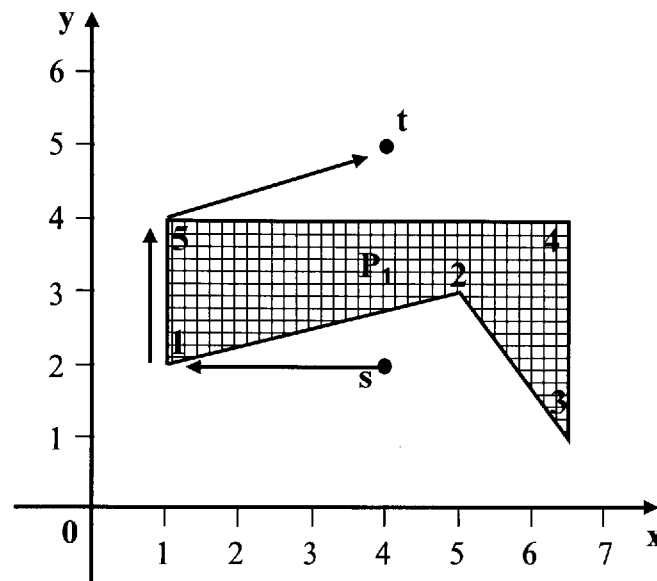


Figure 4.14 The arrows illustrate the semi-free path proposed as a solution to the scenario of Figure 4.13 by the V*MECHA algorithm.

This path is semi-free and is also the shortest path between s and g . The fact that the V*MECHA algorithm discovers the correct solution for Conn *et al* (1997)'s counterexample while the V*GRAPH fails, is an indication (but not a guarantee) that the algorithm works better than the V*GRAPH at least for this particular example. However, a formal proof of the V*MECHA's correctness will be discussed in section 4.8.

Note that since the V*MECHA algorithm uses the A* algorithm to search for the shortest path, the proofs of the V*MECHA's admissibility and optimality are similar to that of the A* algorithm presented in (Hart *et al*, 1968).

4.8 The Admissibility of the V*MECHA Algorithm

For the V*MECHA algorithm to be admissible it has to be proved that it always terminates by returning the shortest semi-free path between the AGV's start point s and its goal point g in the RVG, providing there is one, otherwise it terminates with failure. Having proven Proposition 4.1 and Proposition 4.2 it is ensured that the algorithm will not fail to find a path from s to g , providing that there is one, due to the rejection of the non-super-extremes and the concave vertices. It will be shown that if the V*MECHA algorithm is applied to finite δ graphs and \hat{h} underestimates h , it always finds the shortest paths.

Lemma 4.2

If $\forall n, \hat{h}(n) \leq h(n)$ and the V*MECHA algorithm has not terminated, then there is always a vertex n' on OPEN and on the shortest path, such that $\hat{f}(n') \leq f(s)$, ($f(s)$ is the shortest unconstrained path s to g).

Proof

Let the sequence Path = (s, n_1, n_2, \dots, g) be the optimal path from s to g . At any time before the algorithm terminates let n'' be the vertex last expanded by the algorithm and n' the vertex with the smallest \hat{f} value on OPEN (the successor of n'' on Path), note

that n' could be g . It is known by the definition of \hat{f} that, $\hat{f}(n') = \hat{g}(n') + \hat{h}(n')$, but since n' all its ancestors have been expanded, the optimal path to n' must have been found, so $\hat{g}(n') = g(n')$ therefore,

$$\hat{f}(n') = g(n') + \hat{h}(n') \quad (4.3)$$

and since it is assumed that, $\hat{h}(n') \leq h(n')$ then,

$$\hat{f}(n') \leq g(n') + h(n') = f(n') \quad (4.4)$$

It is known that the value of the function f for any vertex n on the optimal path is equal to $f(s)$, therefore the inequality 4.4 becomes,

$$\hat{f}(n') \leq f(s), \forall n \in \text{Path} \quad (4.5)$$

and the Lemma is proven.

Theorem 4.2

In a finite δ graph, if for every vertex $\hat{h}(n) \leq h(n)$ then the V*MECHA algorithm is admissible.

Proof

Suppose the contrary, that the algorithm terminates without returning the shortest path between s and g . There are three different cases where this could happen.

1st case: The V*MECHA algorithm terminates with a path to a non-goal vertex.

The proof of this contradiction is very trivial and it immediately follows from the termination condition in the algorithm: **if $w = g$ then exit while loop.**

2nd case: The V*MECHA algorithm does not terminate at all.

It is not hard to show that the algorithm always terminates. If the V*MECHA algorithm does not terminate, it is because it expands vertices on OPEN forever and therefore will

expand vertices in the search tree further than $\frac{f(s)}{\delta}$ steps from s . Since the graph is a

finite graph and is also a δ graph, the \hat{g} values of vertex n on OPEN further than $\frac{f(s)}{\delta}$

steps from s in the search tree and thus their \hat{f} values will exceed $f(s)$. However, no

vertex further than $\frac{f(s)}{\delta}$ from s is expanded, because it is known by Lemma 4.2 that

there is some vertex n' on the optimal path such that $\hat{f}(n') \leq f(s)$, therefore the algorithm will expand n' rather than n . The only way that the algorithm can now fail to

terminate is if it keeps reopening vertices within $\frac{f(s)}{\delta}$ steps of s . However, each such

vertex can be opened only a finite number of times since there are finite numbers of

paths for s to n through such vertices within $\frac{f(s)}{\delta}$ steps of s . Therefore the V*MECHA

algorithm terminates.

3rd case: The V*MECHA algorithm terminates with a path to g , however the path is not the shortest.

Suppose that the algorithm terminates at the goal point g but not via the shortest path. From Lemma 4.2, it is known that just before the termination of the algorithm a vertex n' on OPEN and the optimal path existed. Therefore vertex n' would have been chosen for expansion rather than vertex g , which contradicts the assumption that V*MECHA terminated.

Having proved all three cases, it is shown that for finite δ graphs, if $\hat{h}(n) \leq h(n)$ then the V*MECHA algorithm is admissible.

4.9 The Optimality of the V*MECHA Algorithm

As was mentioned in section 4.8, when the encountered graph is a finite δ graph and \hat{h} underestimates h , then the V*MECHA algorithm is admissible. However, there can be several values of \hat{h} underestimating h . For instance if $\hat{h}(n) = 0$ it is still an underestimate of $h(n)$. Note that if $\hat{h}(n) = 0$ the V*MECHA algorithm is a uniform-cost search known as the Dijkstra's shortest path algorithm, (Dijkstra, 1959).

Consider two different versions of the V*MECHA algorithm say V*MECHA₁ and V*MECHA₂ each of them using a different \hat{h} value, say \hat{h}_1 and \hat{h}_2 respectively. If for all non-goal vertices n , $\hat{h}_1 < \hat{h}_2$, it is then said that V*MECHA₂ algorithm is *more informed* than the V*MECHA₁ algorithm.

It will be shown here that the V*MECHA algorithm is optimal in the sense that it never expands more vertices than any other admissible algorithm, which is less than or equally informed as the V*MECHA algorithm, under a restriction on the \hat{h} . The restriction on \hat{h} , is that the difference between the estimated costs from any two visible vertices to the goal vertex g is less than or equal to the cost of the arc connecting the two vertices. More formally for any two visible vertices n_i and n_j ,

$$\hat{h}(n_i) - \hat{h}(n_j) \leq d_{\text{edge}}(n_i, n_j) \quad (4.6)$$

This is called the *consistency assumption* because it must be true when the heuristic information is applied consistently to all vertices.

Lemma 4.3

If the consistency assumption is satisfied then for every expanded vertex n by V*MECHA $\hat{g}(n) = g(n)$. In other words, this Lemma states that if the consistency assumption is satisfied the V*MECHA algorithm has found the optimal path to any vertex n it selects for expansion.

Proof

Suppose that the contrary is true and just before expanding n , that $\hat{g}(n) > g(n)$. Therefore V*MECHA has not found the shortest path, but from Lemma 4.2 it is known that just before the expansion of n , a vertex n' on OPEN and on Path with $\hat{g}(n') = g(n')$ exists. If $n' = n$ then the Lemma holds, otherwise,

$$g(n) = g(n') + d_{\text{edge}}(n', n) = \hat{g}(n') + d_{\text{edge}}(n', n) \quad (4.7)$$

Under the initial assumption $\hat{g}(n) > g(n)$, so it is obtained that

$$\hat{g}(n) > \hat{g}(n') + d_{\text{edge}}(n', n) \quad (4.8)$$

If the $\hat{h}(n)$ is added to both sides of the inequality (4.8), then

$$\hat{g}(n) + \hat{h}(n) > \hat{g}(n') + d_{\text{edge}}(n', n) + \hat{h}(n) \quad (4.9)$$

By applying the consistency assumption to the right hand side of inequality (4.9) it is obtained,

$$\hat{g}(n) + \hat{h}(n) > \hat{g}(n') + \hat{h}(n') \quad (4.10)$$

Therefore, $\hat{f}(n) > \hat{f}(n')$, which contradicts the fact that the algorithm chose n for expansion while n' was available. Therefore the Lemma is proven.

Lemma 4.4

For any vertex n expanded by the V*MECHA, if \hat{h} underestimates h then $\hat{f}(n) \leq f(s)$.

Proof

If the vertex n is expanded by the V*MECHA algorithm and n is the goal g then $\hat{f}(n) \leq f(s)$ because the V*MECHA algorithm is admissible and the Lemma holds. If n is not the goal then it is known from Lemma 4.2 that just before n was expanded there was a vertex n' on OPEN and on Path such that $\hat{f}(n') \leq f(s)$. If $n = n'$ then the Lemma holds otherwise since the algorithm chose n for expansion instead of n' it must have been that,

$$\hat{f}(n) \leq \hat{f}(n') \leq f(s) \quad (4.11)$$

and the Lemma is proven.

Theorem 4.3

If the V*MECHA algorithm is more informed than another admissible algorithm say, MECHA, and the consistency assumption is satisfied, then if a vertex is expanded by V*MECHA, it is also expanded by MECHA.

Proof

Again suppose the contrary, namely that a vertex n expanded by the V*MECHA algorithm but not by MECHA. This could happen because some information should be available to MECHA that the cost of a path through n is equally or more expensive than the shortest path, so $f(n) \geq f(s)$.

It is known that the actual shortest path from s to g constrained to pass through n is $f(n) = g(n) + h(n)$, rearranging this equation it yields,

$$h(n) = f(n) - g(n) \quad (4.12)$$

By the initial assumption it is known that MECHA considered that $f(n) \geq f(s)$, therefore

$$h(n) \geq f(s) - g(n) \quad (4.13)$$

The MECHA algorithm could use as a lower heuristic estimate $\hat{h}(n) = f(s) - g(n)$, while the V*MECHA algorithm uses as a heuristic estimate $\hat{h}(n) = \hat{f}(n) - \hat{g}(n)$, but from Lemma 4.4 it is known that $\hat{f}(n) \leq f(s)$, therefore $\hat{h}(n) \leq \hat{f}(s) - \hat{g}(n)$ meaning that the heuristic function of V*MECHA has satisfied $\hat{h}(n) \leq f(s) - \hat{g}(n)$. However from Lemma 4.3, it is known that $\hat{g}(n) = g(n)$, therefore $\hat{h}(n) \leq f(s) - g(n)$. This indicates that the MECHA algorithm used information on vertex n , which allowed a lower bound for h , at least as large as the one used by the V*MECHA algorithm, contradicting the initial assumption that the V*MECHA algorithm is more informed than the MECHA algorithm. Therefore the theorem holds.

4.10 The Optimality of the Path Produced by the V*MECHA

Algorithm

According to Theorem 4.1 the shortest semi-free path between the AGV's start point s and its goal point g , in an environment populated by static simple polygonal obstacles bends only at obstacles' vertices. Therefore the visibility graph of the environment encapsulates the shortest semi-free path from s to g . From Propositions 4.1 and 4.2 it is

known that the RVG produced by V*GRAPH also encapsulates the shortest semi-free path from s to g . Since the V*MECHA algorithm is admissible it guarantees to find the shortest path within the RVG and therefore the global shortest semi-free path from s to g in the AGV's environment.

4.11 Time and Space Complexities of the V*MECHA Algorithm

In this section an empirical analysis of the computational time and space of the V*MECHA algorithm will be presented.

Theorem 4.4

The V*MECHA algorithm establishes the shortest semi-free path for an AGV, between two query points, in $O(k n \log n)$ computational time and in $O(k^2)$ space, where n is the total number of the obstacles' vertices and k is the total number of the obstacles' non-concave vertices.

Proof

The **while** loop of the algorithm is traversed at most k times since the length of $Open$ is $O(k)$, where k is the total number of the obstacles' non-concave vertices. Therefore there are $O(k)$ iterations and in each iteration the following steps are executed. The identification of the vertex in $Open$ with the smallest \hat{f} value requires $O(k)$ time, where k is the total number of the obstacles' non-concave vertices. The identification of all the w -visible vertices requires $O(n \log n)$ time, where n is the total number of the obstacles' vertices, see section 4.6.2 for details. The identification of the super-

extremes of the extreme vertices of any w -visible sequence requires $O(n \log n)$ time, where n is the total number of the obstacles' vertices, see section 4.6.2 for details. The treatment of w 's children requires $O(k)$ time, where k is the total number of the obstacles' non-concave vertices.

The most computationally expensive steps inside the **while** loop are the identification of the w -visible vertices and the identification of the super-extremes of the extreme vertices of the w -visible sequences. Both of these steps require $O(n \log n)$ time, where n is the number of the obstacles' vertices and these steps dominate the computational time of each iteration of the **while** loop, which is traversed k times, leading the overall computational time of the V*MECHA algorithm to be in $O(k n \log n)$.

The number of edges of the reduced visibility graph produced by the V*MECHA algorithm is bounded by $\binom{k+2}{2}$. Since $\binom{k+2}{2} = \frac{(k+2)(k+1)}{2}$ which is in $O(k^2)$, the space complexity of the V*MECHA algorithm is in $O(k^2)$.

The worst time and space complexities attained by the V*MECHA occur when it is applied in an environment populated only by convex obstacles and k becomes equal to n leading in a computational time $O(n^2 \log n)$ and $O(n^2)$ space.

4.12 Comparing the V*MECHA Algorithm to other Visibility Graph Approaches

In this section the performance of V*MECHA will be compared with various similar algorithms implemented for the solution of the basic movers' problem and will be critically evaluated.

As was mentioned in section 4.2, Rohnert (1986) proposed an algorithm for solving the basic movers' problem in time $O(f^2 + n \log n)$ after $O(n + f^2 \log n)$ pre-processing, where f is the total number of the environment's obstacles and n is the total number of the obstacles' vertices. Rohnert's algorithm is computationally more efficient than V*MECHA, but its main disadvantage over the V*MECHA algorithm is that its applicability is restricted only to environments populated by convex obstacles while the V*MECHA is applicable to environments populated by both convex and non-convex obstacles.

Several methods for constructing the entire visibility graph in time $O(n^2)$, where n is the total number of the obstacles vertices have been proposed, as described in section 4.2. This result leads the VGRAPH approach to be in $O(n^2)$ computational time, where n is the total number of the obstacles' vertices. Comparing the V*MECHA approach to the VGRAPH approach is not very easy because the computational time of the V*MECHA depends on the type of obstacles populating the environment. Therefore two different measurements will be proposed.

The first is when the V*MECHA attains its worst time complexity, this is when the environment contains only convex obstacles. In this case V*MECHA's computational time is $O(n^2 \log n)$, where n is the total number of the obstacles' vertices while the VGRAPH's computational complexity is $O(n^2)$, where n is the total number of the obstacles' vertices. In this case theoretically the VGRAPH algorithm is computationally more efficient than the V*MECHA, but sometimes in practice V*MECHA is faster due to the fact that it does not construct the entire visibility graph but only part of it, as opposed to VGRAPH, which constructs the entire visibility graph. Since fewer vertices are considered for inclusion in the visibility graph, due to the fact that the non-super-extreme vertices are rejected, the OPEN list is smaller resulting in less executions of the algorithm's **while** loop. Therefore even when the environment is populated only by convex obstacles V*MECHA in reality can sometimes be less time consuming than VGRAPH. Also note that since the visibility graph produced by V*MECHA algorithm is smaller than the entire visibility graph produced by VGRAPH, the search process is faster because its efficiency relies on the number of edges of the considered graph.

In the average case, this is when the environment is populated by both convex and non-convex obstacles, V*MECHA is in practice nearly always computationally more efficient. Intuitively in this particular application the average case analysis seems a more realistic measure because in general the physical environment of a robot is populated by obstacles of various shapes and sizes and are more likely to be (or approximated by) non-convex obstacles rather than 'fine' convex obstacles. Figure 4.15

illustrates an environment presented in (Liu and Arimoto, 1992) for the comparison of the T-graph approach with the VGRAPH approach.

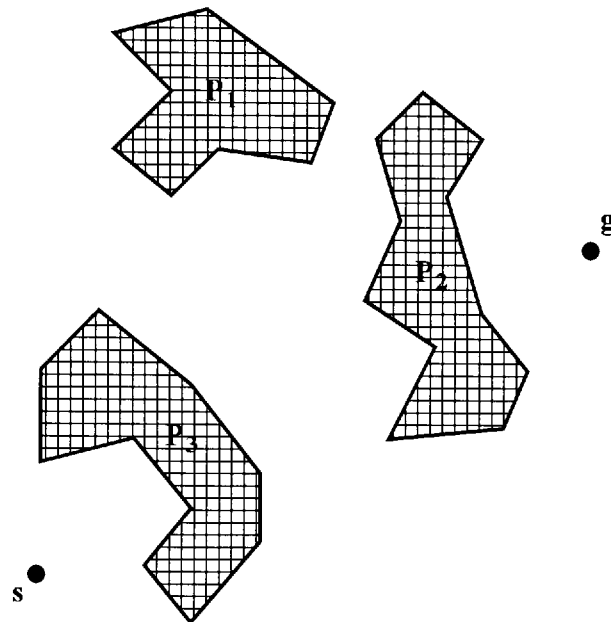


Figure 4.15 The environment used in (Liu and Arimoto, 1992) for the comparison of VGRAPH and T-graph algorithms.

The same environment will be used here in order to demonstrate the large reduction in size of the visibility graph achieved by the V*MECHA algorithm. Figure 4.16a illustrates the visibility graph constructed by the VGRAPH algorithm and Figure 4.16b illustrates the visibility graph constructed by the V*MECHA algorithm.

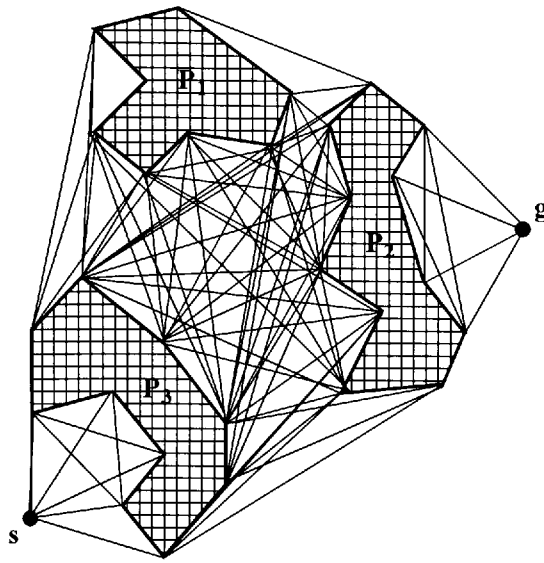


Figure 4.16a Illustration of the visibility graph constructed by the VGRAPH algorithm for the identification of the shortest path between the AGV's start point s and its goal point g .

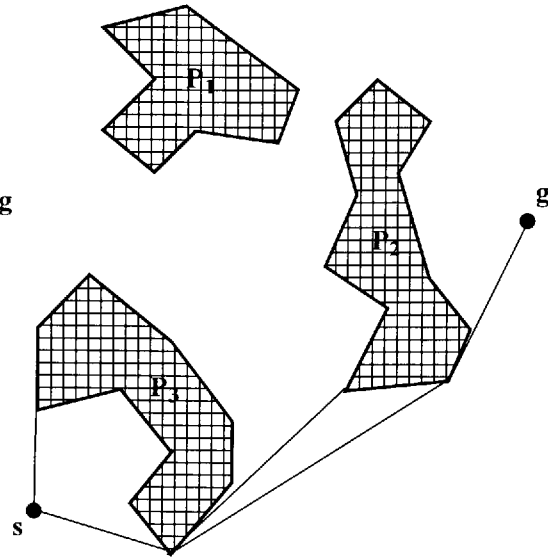


Figure 4.16b Illustration of the visibility graph constructed by the V*MECHA algorithm for the identification of the shortest path between the AGV's start point s and its goal point g .

As can be noticed from Figures 4.16a and 4.16b, the size of the visibility graph produced by the V*MECHA algorithm is significantly smaller than that produced by the VGRAPH algorithm. This reduction makes the search process for the identification of the shortest path much faster.

Even by comparing the V*MECHA algorithm to the T-graph algorithm proposed by Liu and Arimoto (1992), it can be noticed that frequently the visibility graph constructed by the V*MECHA can be smaller in size than the one constructed by the T-graph. Again the same environment will be used to compare the size of the visibility

graph constructed by the V*MECHA to that constructed by the T-graph. Figures 4.17a and 4.17b, illustrate this comparison.

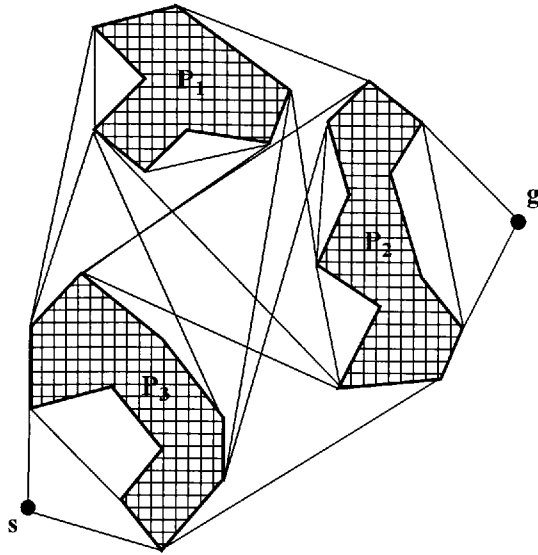


Figure 4.17a Illustration of the visibility graph constructed by the T-graph algorithm for the identification of the shortest path between the AGV's start point s and its goal point g .

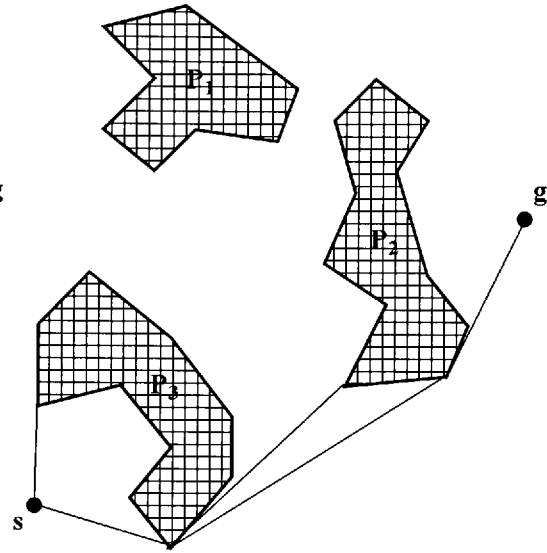


Figure 4.17b Illustration of the visibility graph constructed by the V*MECHA algorithm for the identification of the shortest path between the AGV's start point s and its goal point g .

As can be noticed from Figures 4.17a and 4.17b, the reduction of the size of the visibility graph produced by the V*MECHA algorithm is very apparent in comparison to that of the T-graph approach, thus making the search process for the identification of the shortest path a much faster process. Also, it is worth noting that there are cases when the V*MECHA attains better computational complexity. The computational time of the T-graph approach is in $O(n(n + t) + m^2 t)$, where n is the total number of the

obstacles' convex vertices, m represents the number of the obstacles' convex components and t is the total number of the obstacles' vertices. If the environment is populated by non-convex obstacles with large numbers of concave vertices (i.e. star-like polygonal obstacles), there are lot of convex components in the obstacles which makes the T-graph more expensive computationally and at the same time makes V*MECHA cheaper computationally because there are less vertices for consideration for the construction of the visibility graph and thus less executions of the **while** loop.

4.13 Discussion

In this chapter a new algorithm was proposed for solving the basic movers' problem. The algorithm is called V*MECHA and establishes the shortest semi-free path between two query points for an AGV in a two-dimensional environment which contains simple, convex and non-convex polygonal obstacles. The algorithm is a heuristic search engine and is proposed to correct the V*GRAPH algorithm and overcome its deficiencies.

The computational complexity of the V*MECHA algorithm is empirically shown to be in $O(k n \log n)$, where k is the total number of the obstacles' non-concave vertices and n is the total number of the obstacles' vertices.

Two propositions are presented and proved in order to minimise the number of vertices considered for the construction of the visibility graph without sacrificing the optimality of the path. The significant reduction of the size of the visibility graph enables the

search process to be carried out more efficiently and effectively, since its efficiency is strongly influenced by the number of edges of the considered graph.

The algorithm was proven to be admissible and optimal in a sense that it never expands more vertices than any other less or equally informed admissible algorithm.

Examples showed that the V*MECHA algorithm is superior to those algorithms which construct the entire visibility graph in the average case (which appears most frequently in real world environments) and that it can be faster than the T-graph algorithm in some cases.

In the next chapter the V*MECHA algorithm will be used as the basis for the development of another algorithm called D*MECHA. The D*MECHA algorithm finds the time-minimal semi-free motion between two query points for an AGV, which operates in a two-dimensional environment populated by simple convex and non-convex polygonal obstacles which can be static or moving, following a predefined motion.

4.14 References

- ALEXOPOULOS, C. and GRIFFIN, P. 1992. Path Planning for a Mobile Robot. *IEEE Transactions on Systems, Man and Cybernetics*, **22** (2), pp. 318 – 323.

- ASANO, T., ASANO, T., GUIBAS, L., HERSHBERGER, J. and IMAI, H. 1985. Visibility-Polygon Search and Euclidean Shortest Paths. *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*. Portland, Oregon. pp. 155 - 164.
- BARR, A. and FEIGENBAUM, E. A. 1983. *The Handbook of Artificial Intelligence*. Volume 1. London, U.K: PITMAN BOOKS LIMITED.
- CONN, R. A., ELENES, J. and KAM, M. 1997. A Counterexample to the Alexopoulos - Griffin Path Planning Algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, Part B, **27** (4), 721 – 723.
- DIJKSTRA, E. 1959. A Note on two Problems in Connection with Graphs. *Numerische Mathematik*, **1**, pp. 269 – 271.
- EDELSBRUNNER, H. 1987. *Algorithms in Combinatorial Geometry*. EATCS Monographs on Theoretical Computer Science. Volume 10. Springer – Verlag.
- FLEISCHER, R., FRIES, O., MEHLHORN, K., MEISER, S., NÄHER, S., ROHNERT, H., SCHIRRA, S., SIMON, K., TSAKALIDIS, A. AND UHRIG, K. 1992. Selected Topics from Computational Geometry, Data Structures and Motion Planning. *Lecture Notes in Computer Science*, 594, pp. 25 – 43.

- FREDMAN, M. and TARJAN, R. 1984. Fibonacci heaps and their uses in improved network optimisation algorithms. *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*. pp. 338 – 346.
- HART, P. E., NILSSON, N. J. and RAPHAEL B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions of Systems Science and Cybernetics*, **4** (2), pp. 100-107.
- JIANG, K, SENEVIRATNE, L. D. and EARLES, S. W. E. 1996. Motion Planning with Reversal Manoeuvres for a Non-Holonomic Constrained Robot. *Proceedings ImechE, Journal Engineering Manufacture*, **210** (5), pp. 487 - 497.
- JIANG, K, SENEVIRATNE, L. D. and EARLES, S. W. E. 1999. A Shortest Path Based Path Planning Algorithm for Nonholonomic Mobile Robots. *Journal of Intelligent and Robotic Systems*, **24**, pp. 347 - 366.
- LEE, D. T. 1978. *Proximity and Reachability in the Plane*. Ph.D. Thesis, Coordinated Science Laboratory, University of Illinois. Urbana, IL.
- LIU, Y. and ARIMOTO, S. 1991. Proposal of Tangent Graph and Extended Tangent Graph for Path Planning of Mobile Robots. *Proceeding of the 1991 IEEE International Conference on Robotics and Automation*. Sacramento - California. pp. 312 - 317.

- LIU, Y. and ARIMOTO, S. 1992. Path Planning Using a Tangent Graph for Mobile Robots Among Polygonal and Curved Obstacles. *The International Journal of Robotics Research*, **11** (14), pp. 376 - 382.
- LIU, Y. and ARIMOTO, S. 1995. Finding the Shortest Path of a Disc Among Polygonal Obstacles Using a Radius-Independent Graph. *IEEE Transactions on Robotics and Automation*, **11** (5), pp. 682 - 691.
- LOZANO-PÉREZ, T. and WESLEY, M. A. 1979. An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles. *Communications of the ACM*, **22** (10), pp. 560 - 570.
- NILSSON, N. J. 1969. A Mobile Automaton: an Application of Artificial Intelligence Techniques. *Proceedings of the 1st International Joint Conference on Artificial Intelligence*. Washington D.C. pp. 509 - 520.
- NILSSON, N. J. 1980. *Symbolic Computation, Principles of Artificial Intelligence*. Los Altos, CA: Morgan Kaufmann.
- NILSSON, N. J. 1998. *Artificial Intelligence: A New Synthesis*. San Francisco, Morgan Kaufmann Publishers, Inc.
- PEARL, J. 1983. Knowledge Versus Search: A Quantitative Analysis Using A*. *Artificial Intelligence*, **20**, pp. 1 – 13.

- PEARL, J. 1984. *Heuristics, Intelligent Search Strategies for Computer Problem Solving*. Addison – Wesley Publishing Company.
- PREPARATA, F. P. and SHAMOS, M. I. 1985. *Computational Geometry: An Introduction*. New York: Springer-Verlag.
- ROHNERT, H. 1986. Shortest Paths in the Plane with Convex Polygonal Obstacles. *Information Processing Letters*, **23**, pp. 71 - 76.
- ROHNERT, H. 1988. Time and Space Efficient Algorithms for Shortest Paths Between Convex Polygons. *Information Processing Letters*, **27**, pp. 175 - 179.
- WELZL, E. 1985. Constructing the Visibility Graph for n-Line Segments in $O(n^2)$ Time. *Information Processing Letters*, **20**, pp 167 - 171.

5

The D*MECHA Algorithm for Motion Planning of an AGV in Dynamic Environments

For every matter has its time and method

ECCLESIASTES 8: 6

5.1 Introduction

In this chapter the problem of planning the motion of an AGV in environments populated with both static and moving obstacles is considered and an extension of the V*MECHA algorithm for solving this problem is proposed. The two-dimensional dynamic motion planning problem is often referred to in the literature as the *two-dimensional asteroid problem* or the *cocktail party problem*.

The trajectories of the obstacles are supposed to be known or easily computed as a function of time. Note that in static environments the solution of the path planning problem is just about spatial reasoning, while in dynamic environments its solution should incorporate the parameter of time. Therefore in dynamic environments, where the AGV's path should be defined as a function of time the term motion is used instead of path and hence motion planning instead of path planning. More formally the dynamic motion planning problem is posed as follows.

Problem 2

Consider the problem of planning the motion of an AGV R in a two-dimensional workspace W populated by simple polygonal obstacles P_i , where $i \in \mathbb{N}$, the AGV's start location s and its goal location g . The AGV is a point-robot, which translates freely at fixed orientation with bounded velocity modulus. The maximum velocity that the robot can reach is denoted by \vec{v}_{\max_R} . Every P_i in W can be static or moving along linear paths at fixed orientation, with constant velocity, which is less than \vec{v}_{\max_R} . Every moving P_i , before and after its motion has velocity equal to zero. The environment's obstacles are not allowed to come into contact with each other at any time. The problem is to plan a time-minimal semi-free motion for R , from its start point to its goal point, given that the AGV's start and goal points are collision-free at all times and that the descriptions of the obstacles (such as shapes, locations and velocities) are accurately known ahead of planning.

A number of mild assumptions are considered in order to make the problem more tractable. These assumptions are the following:

1. The obstacles are not allowed to come in contact with each other at any time. This assumption simply maintains the topology of the free space the same at all times.
2. The robot's start and goal points are collision-free at all times. This assumption simplifies the solution of the problem by avoiding extra checking at the start and the end of the robot's motion.
3. It is supposed that the moving obstacles and the robot can accelerate and decelerate instantly. This assumption makes the solution to the problem simpler by not having to consider acceleration and deceleration constraints.
4. The velocity of every obstacle is smaller than the velocity of the robot. Under the premises of this assumption the time-minimality theorem is realisable (see section 5.8 for details).

Some of these assumptions can be relaxed however at the expense either of the computational time of the algorithm or of the minimality of the produced motion or even of the solutions completeness.

The problem considered in this chapter is far more complicated than the problem considered in chapter four whose domain was a stationary environment, but at the same time is more realistic. The reason for this is that most of the environments where an AGV operates are reconfigured over time regardless of whether they are indoor, such as industrial environments where other robots or humans co-exist or outdoors where

physical obstacles are in motion. In section 5.2 a number of studies are overviewed whose results indicate that the robot motion planning in time varying environment is intrinsically harder problem than the robot path planning among stationary obstacles. However, there are algorithms, which work in limited domains (problem domains under mild assumptions) and attain polynomial time (Fujimura, 1991). Here an algorithm will be proposed for solving the motion planning problem in polynomial time for an AGV which operates in a time-varying environment under the mild assumptions considered by the specification of the problem. This algorithm is an extension of the V*MECHA algorithm presented in chapter four and is called D*MECHA.

5.2 Related Work

Sutner and Maass (1988) considered the motion planning problem for a point-robot with bounded velocity modulus among dynamic obstacles in one-dimension. In their approach they introduced another dimension into the configuration space, so the one-dimensional moving obstacles were described as polygonal objects in space-time. The motion planning of a point-robot with bounded speed in one dimension was solved in polynomial time on the total number of vertices of the polygonal space-time obstacles.

In (Canny and Reif, 1987) it was shown that motion planning for a point-robot, with a bounded velocity modulus, in a two-dimensional environment populated by arbitrarily many moving, non-rotating convex obstacles, that move at constant velocity is NP-hard.

Kant and Zucker (1986), decompose the trajectory planning problem (TPP) into two sub-problems. (i) The path planning problem (PPP) in which a path that avoids collisions with static obstacles, is planned and (ii) the velocity planning problem (VPP) in which the velocity that avoids collisions with moving obstacles along this path, is planned. The VPP is posed in path-time space where time is explicitly represented as an extra dimension reducing the problem to a graph search leading to the transformation of the VPP into a PPP. The limitation of this approach is that the AGV is not allowed to alter the path established in the path planning stage, but only its velocity along this path. This means that the algorithm will not find a solution when one exists for the case where an obstacle is moving along the path of the AGV.

Erdmann and Lozano-Pérez (1987) consider the problem of planning motion for multiple robots. They assign priorities to each robot and then they plan the motion of one robot at a time according to its priority. The space-time configuration space was represented as a list of configuration space slices at particular points in time. These times are those at which a moving object changes its velocity. A motion consists of a series of straight-line segments each of them starting at a node of a slice and terminating at a node of the next slice. The robot follows a straight-line motion with constant velocity between the nodes of two slices. Along the path the robot changes its velocity only at nodes of obstacles, when some obstacle's velocity changes. Due to the discretisation of the time the algorithm might fail to find a path because it only generates a few path segments. The authors alleviated this problem by introducing extra configuration space slices between those already represented, at the expense of the algorithm's computational time. The algorithm runs in time $O(r n^3)$, where n is the total

number of the environment's edges and r is the total number of the constructed slices. The algorithm is time resolution-complete between the slices. A way to make the algorithm complete is to introduce slices at the times where the topology of the free space changes.

Fujimura and Samet consider the dynamic robot motion planning problem and reported a number of papers considering different types of environments. A summary of these results can be found in (Fujimura, 1991). Fujimura and Samet (1993) presented an algorithm to find a motion for a point-robot, in an environment populated by time-dependent obstacles and destination point. The environment's obstacles are convex polygons, which move in a fixed direction at constant speed. The algorithm they proposed finds the time-minimal motion given that the point-robot moves faster than the obstacles and the destination point. This algorithm is based on the concept of the accessibility graph. Accessibility graph is a directed graph with nodes, all the accessible by the robot obstacle's vertices and destination point, as well as the robot's start point. Note that this graph is embedded in the two-dimensional configuration space of the robot and not in its three-dimensional space-time configuration space. This graph can be infinite and therefore the authors considered a finite version of the accessibility graph by considering the accessible points with the youngest accessible time. The algorithm searches this graph and finds the time-minimal motion from the robot's start point to its goal point in $O(n^2 \log n)$ computational time, where n is the total number of the obstacles' vertices. Fujimura (1993) proposed two algorithms for solving the motion planning problem for a point robot that moves in an environment where the moving obstacles and the destination point have a cyclic motion. These

algorithms are the *hit-and-leave*, which is suited to sensor-based navigation and the *accessibility* algorithm, which is more suited when the environment is accurately known ahead of planning. Both of these algorithms establish a collision-free motion (more precisely a semi-free motion), providing that the robot moves faster than the obstacles. The motion defined by the second method is also time-minimal. Fujimura (1994), proposed an algorithm for motion planning of a point-robot in an environment with transient obstacles and destination point. Transient obstacles are those obstacles whose existence in the environment is time-dependent. The algorithm was based on the propagation of a wave-front technique in order to identify the robot's accessible points and then construct an accessibility graph. The time-minimal motion for the robot from the start to the goal point was then established in $O(n^3 \log n)$ computational time, where n is the total number of the environment's vertices.

Reif and Sharir (1985) showed that the problem of motion planning in a three-dimensional environment populated by moving obstacles is a PSPACE-hard problem, when the robot's velocity modulus is bounded and NP-hard when the robot's velocity modulus is not bounded.

5.3 The Space-Time Configuration Space

As was mentioned in the introduction of this chapter (section 5.1), in dynamic environments the solution of the path planning problem is not just about spatial reasoning, because the parameter of time should be taken into consideration. Note that

in such environments a path could be collision-free in a specific period of time t_α and not free of collisions in a different period of time t_β .

Since the position of the environment's obstacles changes over time, the motion of the AGV should be defined as a continuous function of time. This can be achieved by defining the AGV's *space-time configuration space CT*. The AGV's space-time configuration space is defined, by adding the dimension of time in its configuration space C . Since the AGV is a point-robot with two degrees of freedom (the two translational) its configuration space C is identical to its workspace W at the same time instance. Note that the CP_i s are shape invariant under translations and therefore, $\forall i, CP_i = P_i$ at all times. Thereby in time dynamic environments when the AGV only translates, every obstacle's configuration space remains unchanged. The only thing that changes is its position and specifically it moves in the same way as its corresponding obstacle moves in W .

Figure 5.1 depicts a two-dimensional workspace W populated by two obstacles P_1 and P_2 and the AGV's start and goal point. The P_1 obstacle is static while P_2 is moving and moves along the direction indicated between times t_α and t_β . Figure 5.2a illustrates the AGV's configuration space at time t_α and Figure 5.2b illustrates the AGV's configuration space at time t_β .

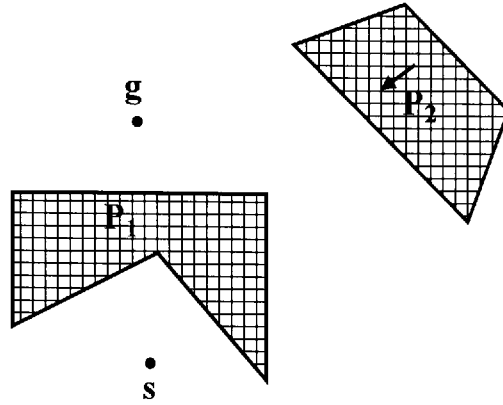


Figure 5.1 The AGV's workspace W populated by a stationary and a moving obstacles as well as the AGV's start and goal points.

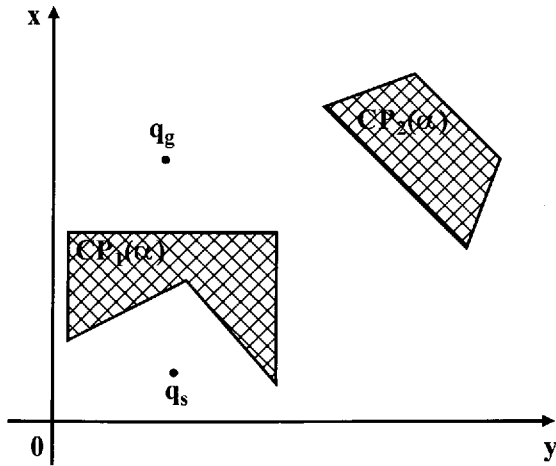


Figure 5.2a The AGV's configuration space C at time t_α .

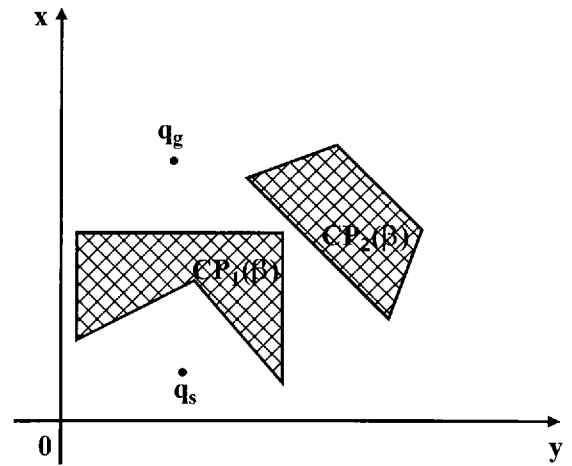


Figure 5.2b The AGV's configuration space C at time t_β .

The AGV's space-time configuration space $CT = C \times [0, +\infty)$ and thus $CT = \mathbb{R}^2 \times [0, +\infty)$. Every CP_i maps from C into CT as a prism, which is denoted by CTP_i . The static CP_i s map into prisms, which are orthogonal to the x - y plane in CT and the time-varying CP_i s map into prisms, which are sloped to the x - y plane in CT . The

angle of the slope of any CTP_i is proportional to the corresponding obstacle's constant velocity.

It is known from the specification of the problem in section 5.1 that the boundaries of the obstacles do not come in contact with each other at any time. Therefore, the boundaries of the CP_is in C and the boundaries of the CTP_is in CT do not come in contact in contact with each other at any time. Figure 5.3 illustrates the AGV's three-dimensional space-time configuration space CT for the environment of Figure 5.1.

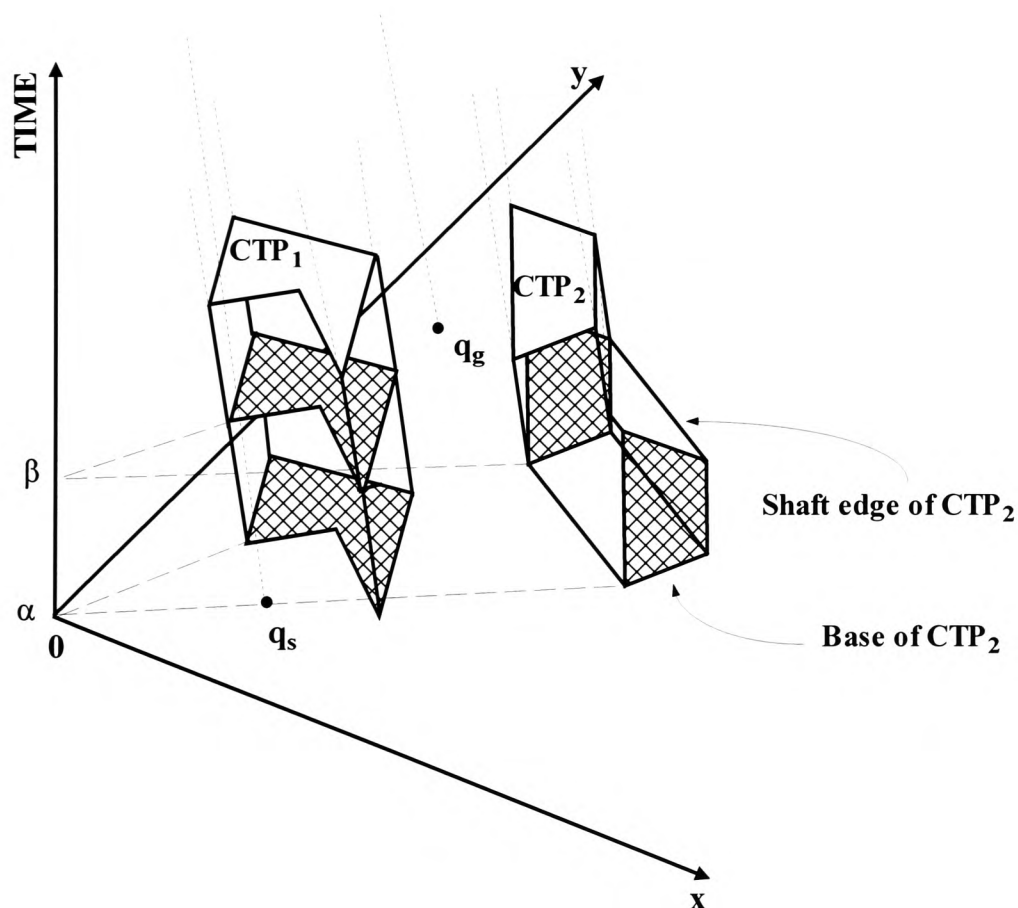


Figure 5.3 The AGV's space-time configuration space CT for the environment of Figure 5.1.

Note that after the end of each obstacle's motion the corresponding prism becomes orthogonal to the x-y plane. As can be noticed from Figure 5.2b and 5.3 all the edges of the CP_i s map into the faces of the prisms (CTP_i s) in CT and all the vertices of the CP_i s map into the edges of the prisms, which do not constitute the bases of them, these edges are called *shaft edges*. The AGV's start and the goal configurations in C correspond to half lines in CT, which emanate from q_s and q_g respectively.

With the construction of the AGV's space-time configuration space, the problem of planning a motion for an AGV in a dynamic environment has been simplified to that of planning a path for an AGV in a static environment. A path in CT encapsulates both time and location information, therefore since the AGV is moving with constant velocity the Euclidean shortest path from q_s to q_g in the three-dimensional CT, corresponds to the time-minimal motion from q_s to q_g in C. Once the space-time configuration space has been constructed, it can then be searched for a collision-free shortest path from q_s to q_g .

5.4 'Reachability' and Visibility

Since the AGV is a point-robot not subject to any kinematic or dynamic constraints, in order to move from its start point to its goal point by attaining a time-minimal semi-free motion it should always move with its maximum velocity \bar{v}_{\max_R} .

A set of all reachable configurations for the AGV when moving with constant velocity (\bar{v}_{\max_R}), from a specific configuration p in C, between two time instances t_α and t_β (with

$t_\alpha < t_\beta$), is bounded by the perimeter of a circle CR, with centre p and radius $r = v_{\max_R} (t_\beta - t_\alpha)$. Figure 5.4 illustrates the set of all reachable configurations for the AGV R in C between t_α and t_β time instances.

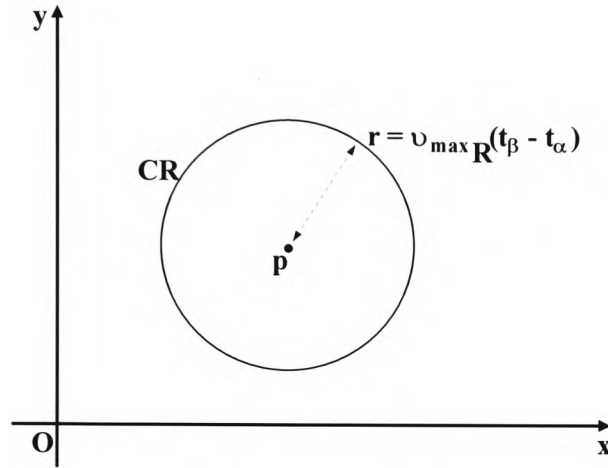


Figure 5.4 All the reachable configurations from p are bounded by the perimeter of the circle CR.

Given that the AGV is moving with constant velocity (\vec{v}_{\max_R}), the set of all the configurations that can reach between two time instances t_α and t_β (with $t_\alpha < t_\beta$) from a single configuration p in CT, is defined by the surface of a right circular cone CN, which emanates from p . Figure 5.4 illustrates the set of all reachable configurations for the AGV in CT between t_α and t_β time instances.

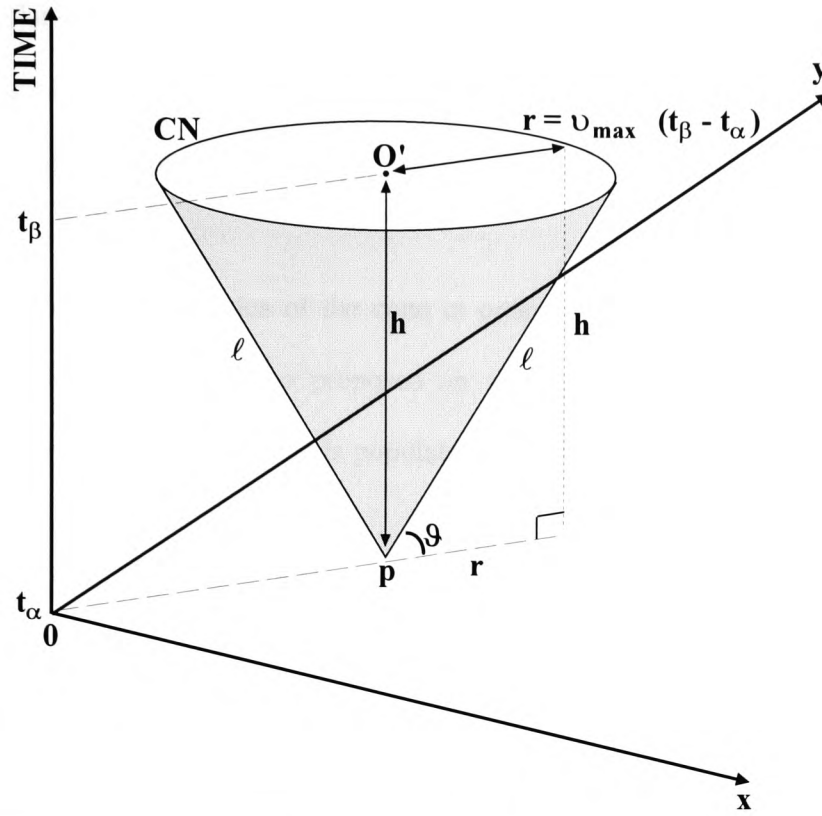


Figure 5.5 All the reachable configurations from p are defined by the surface of the cone CN.

The configuration p is the apex of the cone CN. The height h of CN is parallel to the time axis and is equal to $t_\beta - t_\alpha$. The radius r of the circular base of CN is equal to $v_{\max_R} (t_\beta - t_\alpha)$. The angle θ created by the slant height ℓ of CN and the x-y plane is defined as follows:

$$\tan \theta = \frac{h}{r} \Leftrightarrow \tan \theta = \frac{t_\beta - t_\alpha}{\bar{v}_{\max_R} (t_\beta - t_\alpha)} \Rightarrow \theta = \tan^{-1}(\bar{v}_{\max_R}^{-1}) \quad (5.1)$$

If CT is polar-swept with a half-line emanating from a point p , at angle θ (with the x-y plane), then all the intersections between the half-line and any of the shaft edges of the prisms correspond to reachable configurations by the AGV from p .

Fujimura (1994) used the idea of the cone in order to identify the collision fronts of transient obstacles and then he proposed an algorithm for finding a time-minimal motion for an AGV in environments populated by transient obstacles and a destination point.

Figures 5.6a and 5.6b give some insight into the idea of the cone. Suppose that a cone CN emanates from a configuration, say p on the half-line q_s in CT, with projections on the x-y plane and the time axis, (x_p, y_p) and $t_p = 0$ respectively. Further, suppose that CN intersects the shaft edge 2 of the CTP_1 at configuration say q , with projections on the x-y plane and the time axis, (x_q, y_q) and t_q respectively. This means that the AGV is capable of reaching configuration q from p in time $t_q - t_p$ given that the AGV is travelling with constant speed equal to $v_{\max R}$. This scenario corresponds to configuration space C as follows. If the AGV leaves q_s , (or configuration p with co-ordinates (x_p, y_p)) at the time instance $t_p = 0$, then it is capable of reaching vertex 2 of CP_1 , at a configuration q with co-ordinates (x_q, y_q) at time instance t_q , given that the AGV is moving with constant speed equal to $v_{\max R}$. Then it is said that vertex 2 of CP_1 is reachable from configuration p at configuration q in time t_q , and within time $t_p - t_q$.

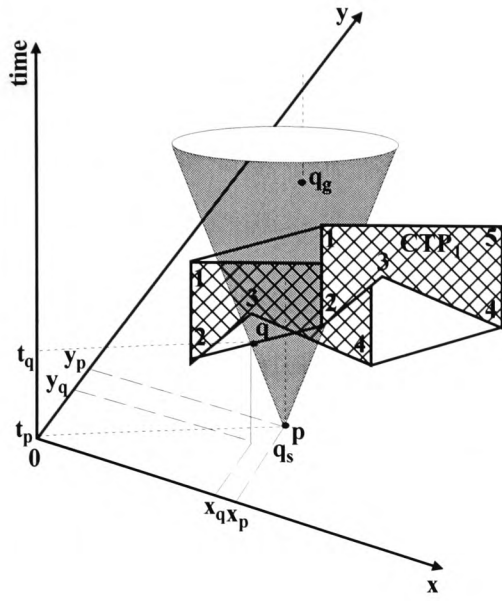


Figure 5.6a 'Reachability' in CT.

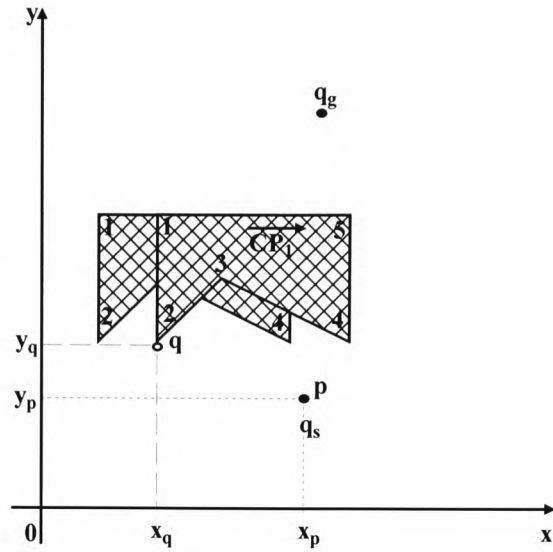


Figure 5.6b 'Reachability' in C.

However, in order to say that an AGV is capable of moving from configuration p to configuration q , 'reachability' is not enough. Another condition, which has to be satisfied, is visibility. This means that the two configurations can be connected with an edge and this edge does not overlap the interior of any CTP_i in CT. In summary, if a ray v is swept about p keeping a constant angle $\vartheta = \tan^{-1}(\bar{v}_{\max_R}^{-1})$ with the x - y plane, all the p -visible configurations at angle ϑ in the CT can be identified. These p -visible configurations at angle ϑ in CT correspond to p -visible vertices in C at a specific location at a specific time instance, which can be reached from p by the AGV, given that it is moving with constant velocity equal to \bar{v}_{\max_R} . In this way a graph demonstrating reachability and visibility ('reacha-visibility graph' RVG_ϑ) at angle ϑ can be constructed in CT and then searched for a path. Note that the RVG_ϑ is a directed graph because every motion of the AGV should be strictly monotone in time.

5.5 Proposition of the D*MECHA Algorithm

As was mentioned in the introduction of this chapter (section 5.1) the D*MECHA algorithm is an extension of the V*MECHA algorithm presented in chapter four. Therefore the two propositions used by the V*GRAPH algorithm in order to minimise the number of the obstacles' vertices considered for the construction of the visibility graph will also be used by the D*MECHA algorithm. However, note that even though the propositions are the same, their validity when applied in time-varying environments has to be proved. Note also that the Theorem 4.1 presented in chapter four, is applicable in dynamic environments as well, therefore more formally Theorem 5.1 is stated as follows.

Theorem 5.1

Given the AGV's start point s , its goal point g and a set P of moving and stationary simple polygonal obstacles, the time-minimal semi-free motion from s to g (providing that there is a motion from s to g), turns only at obstacles' vertices.

Proof

The proof of the theorem is similar to that of Theorem 4.1 in chapter four and it follows immediately once the AGV's space-time configuration space CT is constructed.

Theorem 5.1 entails that the time-minimal motion between two given points in an environment populated by moving and static polygonal obstacles is contained in the RVG_0 in CT . Therefore the minimum-cost path in RVG_0 in CT between nodes q_s and q_g corresponds to time-minimal motion in the environment between the AGV's start and

goal points. However, notice that in dynamic environments an obstacle's vertex (a shaft edge of a CTP_i in CT) can be visible and reachable from different vertices more than once, in different locations and different times. Thus resulting in an arbitrarily large RVG_9 in CT and hence making the search process of the RVG_9 extremely time consuming. A way to alleviate this problem is by using Lemma 5.1.

For the sake of the simplicity, Lemma 5.1, Lemma 5.2 and Propositions 5.1 and 5.2 (see below) won't be proved in the three-dimensional CT, but in the two-dimensional configuration space C, taking the parameter of time into consideration.

Lemma 5.1

If a vertex of an obstacle (a shaft edge in CT) is considered more than once for the construction of the RVG_9 in CT then the corresponding configuration with the youngest reachable time should be retained.

Proof

Recall from section 5.4 that in order for the AGV to get from q_s to q_g by following the time-minimal semi-free motion, it should always move with its maximum velocity. Now suppose that the vertex v of a C-Obstacle (shaft edge v in CT) is visible and reachable from a configuration q_1 , at a configuration q_v and time t_v . Suppose further that as the algorithm proceeds, vertex v is also visible and reachable by a configuration q_2 , at configuration q_v' and time t_v' , with $t_v' > t_v$. Since the AGV can be at configuration q_v at time t_v , if it gets hooked on vertex v and follows the motion of the C-Obstacle it can be at configuration q_v' at time t_v' , however by moving with velocity less

than \bar{v}_{\max_R} (recall that according to the forth assumption the velocity of every obstacle is less than \bar{v}_{\max_R}). Since in the time-minimal semi-free motion the AGV should move at its maximum velocity the motion through configuration q_v' is not time-minimal. Therefore if a vertex is visible and reachable more than once, the configuration with the youngest reachable time should be considered for the construction of the RVG_g . Note that vertices of the RVG_g are not the original C-Obstacles' vertices as in the case with static environment (chapter four) but the configurations where these vertices are visible and reachable. The importance of this Lemma is that it maintains the number of vertices of the $RVG_g \leq n + 2$, where n is the total number of the C-Obstacles vertices.

Before Proposition 5.1 and Proposition 5.2 that are used for the reduction RVG_g size, are discussed, Lemma 5.2 should be presented.

Lemma 5.2

A time-minimal semi-free motion visits only configurations in CT, which correspond to extreme vertices of a visible sequence in C.

Proof

Consider the AGV's configuration space C of Figure 5.7, the C-Obstacle CP_1 is moving in the direction indicated. There are two different ways for the AGV to reach q_g from q_s . The first is to move directly to the q_g , stop and wait at configuration q_α until the C-Obstacle moves out of its way and then carry on moving towards q_g . The second is to go around either side of the C-Obstacle. It will be proved in section 5.8 that the former way of getting to q_g is not time-minimal.

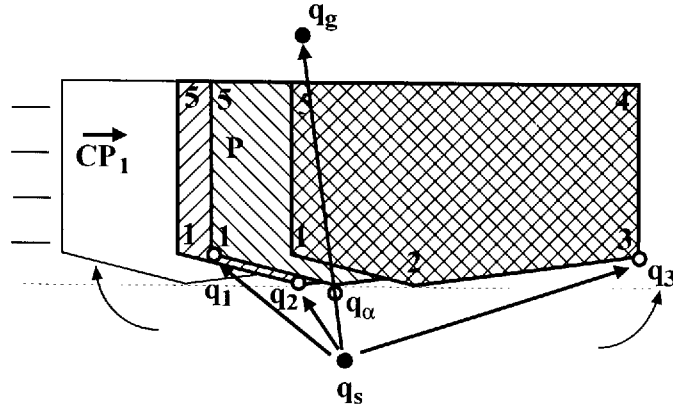


Figure 5.7 The time-minimal motion does not visit non-extreme vertices of the visible sequence.

Therefore the AGV will go around CP_1 , either from the left hand side or the right hand side, in order to achieve a time-minimal motion. Suppose that the time-minimal semi-free motion is the one, which goes around the left hand side of the CP_1 , further suppose that vertices 1, 2, 3 are all reachable and visible from q_s at configurations q_1 , q_2 and q_3 and time t_1 , t_2 and t_3 respectively. Vertices 1, 2, 3 are reachable, visible from q_s and they are consecutive on a single C-Obstacle's boundary therefore, they comprise a q_s -visible sequence. Since the time-minimal motion is the one, which goes around the left-hand side of the CP_1 the AGV will definitely pass through vertex 1, while moving along the time-minimal semi-free motion. Therefore for the validity of the lemma it has to be shown that the straight line motion from q_s to vertex 1 at configuration q_1 is less time consuming than any other polygonal motion from q_s to vertex 1 through a non-extreme vertex of the q_s -visible sequence, which meets vertex 1 at a configuration q_1' .

Suppose the contrary, namely the time-minimal motion visits non-extreme vertices of visible the sequence and that in the configuration space of Figure 5.7 the time-minimal

motion from q_s to vertex 1 goes through vertex 2, which is the non-extreme vertex of the q_s -visible sequence $\{1, 2, 3\}$.

If the AGV departs from q_s at time t_0 it can reach vertex 1 at configuration q_1 at time t_1 , by moving on the straight-line segment $\overline{q_s q_1}$. If the AGV departs from q_s at time t_0 it will eventually reach vertex 2 at configuration q_2 at time t_2 and if it starts moving from configuration q_2 towards vertex 1 at time t_2 , it will eventually reach vertex 1 at a configuration q_1' and time t_1' . Since it is assumed that the motion from q_s to vertex 1 through vertex 2 is time-minimal then it must hold that,

$$t_1' = t_2 + (t_1' - t_2) \leq t_1 \quad (5.2)$$

This means that after the AGV departs from q_2 at t_2 it can be at q_1 either at time $t_1' = t_1$ (note this is the upper bound of the inequality 5.2) or in time t_1'' with $t_1' < t_1'' < t_1$ (note that if the AGV is at q_1 at time t_1'' , vertex 1 of the C-Obstacle has not arrived yet in configuration q_1). However, it is well known the straight-line segment, which connects two points (in this case q_s and q_1) is shorter than any other polygonal line with the same endpoints and since the AGV moves with constant velocity (equal to \vec{v}_{\max_R}), motion $\{q_s, q_1\}$ is less time consuming than motion $\{q_s, q_2, q_1\}$ and therefore, $t_1 < t_1'$, which contradicts inequality 5.2 and in turn the initial assumption that the time-minimal semi-free motion visits non-extreme vertices of the q_s -visible sequence. This proves the Lemma.

Proposition 5.1

A time-minimal semi-free motion only visits configurations in CT, which correspond to super-extremes of the extreme vertices of the visible q_s -sequences for each CP_i in C.

Proof

According to Lemma 5.2 the time-minimal semi-free motion visits only configurations in CT, which correspond to extreme vertices of the visible sequences in C. Using the same arguments as in Lemma 5.2 it can be shown that the time-minimal semi-free motion never visits configurations in CT, which correspond to the non-super-extremes of the extreme vertices of the visible sequences for each CP_i in C.

As with the V*MECHA algorithm, when the D*MECHA identifies the super-extremes of the q_s -visible sequences, it marks all the non-super-extremes as useless. In this way they are not considered again later by the algorithm, if they are visible from a different vertex, say r , regardless whether they are super-extremes or non-super-extremes of the r -visible sequences. The reason for rejecting permanently the non-super-extremes is as follows.

Suppose that vertex, say v , is a non-super-extreme vertex of the extremes of the k -visible sequences for the CP_1 and is reachable at configuration q_v at time t_v . If v is visible and reachable from another vertex later in the algorithm, say vertex r (which is accessible from k), at configuration q_v' at time t_v' then if v is non-super-extreme of the extreme vertices of the r -visible sequences for CP_1 it will be rejected. If v is a super-extreme of the extreme vertices of the r -visible sequences for CP_1 then it does need to

be considered for the construction of the visibility graph because $t_v < t_v'$ and according to Lemma 5.1 the configuration q_v is retained because it has smaller reachable time. But since v is a non-super-extreme of the extreme vertices of the k -visible sequences for CP_1 , it does not need to be considered for the construction of the visibility graph. Therefore once the non-super-extremes of the extreme vertices of the visible sequences for each CP_i are identified, they can be rejected permanently.

Proposition 5.2

Configurations in CT, which correspond to concave vertices of non-convex C-Obstacles, should not be included in the time-minimal semi-free motion.

Proof

Consider the AGV's configuration space of Figure 5.8, the C-Obstacle CP_1 is moving in the direction indicated. There are two different ways for the AGV to reach q_g from q_s . The first is to move directly to the q_g , stop and wait at configuration q_α until the C-Obstacle moves out of its way and then carry on moving towards q_g . The second is to go around either side of the C-Obstacle (top or bottom side). It will be proven in section 5.8 that the former way of getting to the q_g is not time-minimal.

Assume that the time-minimal motion is the one, which goes around the bottom side of the CP_1 and specifically is the motion $\{q_s, q_3, q_4, q_5, \text{ and } q_g\}$. Note that this motion goes via the concave vertex 4 of CP_1 .

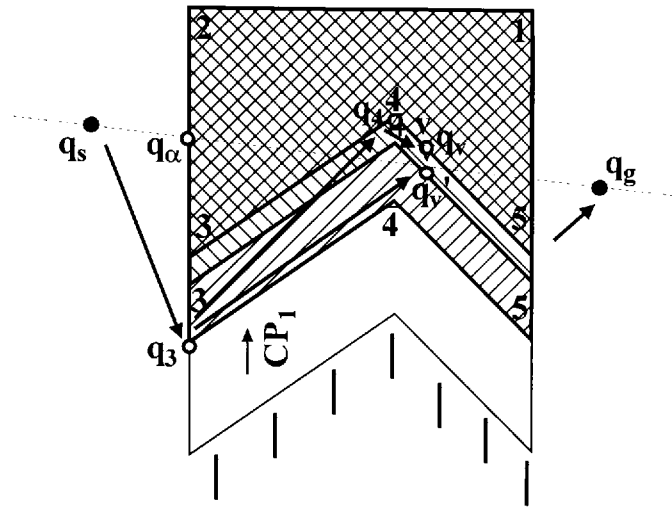


Figure 5.8 The time-minimal motion does not visit concave vertices.

If the AGV starts moving from q_s at time t_0 it can reach vertex 3 at configuration q_3 at time t_3 . If it starts moving from q_3 at time t_3 it can reach vertex 4 at configuration q_4 at time t_4 . Recall from the specification of the problem that the obstacles are not allowed to come in contact with each other at any time, this means that there should be at least a sufficiently small clearance distance σ in between them. Since vertex 4 is visible and reachable from q_3 and CP_1 is a simple polygon then there is always a point v between vertex 4 and its successor within distance σ from 4 which is also visible from q_3 . Point v is reachable and visible from q_4 at configuration q_v and time t_v . Point v is also reachable from q_3 at configuration $q_{v'}$ and time $t_{v'}$, with $t_{v'} < t_v$, because the straight line motion $\{q_3, v'\}$ is less time consuming than any other polygonal motion with the same endpoints (in this case $\{q_3, q_4, q_v\}$). Note that motion $\{q_3, q_v\}$ is collision-free, since the AGV always moves within distance σ from CP_1 . Therefore the motion $\{q_s, q_3, q_{v'}, q_5, q_g\}$ is less time consuming than the motion $\{q_s, q_3, q_4, q_5, q_g\}$, which contradicts the assumption that the motion $\{q_s, q_3, q_4, q_5, q_g\}$ is time-minimal. This contradiction leads

to the conclusion that the time-minimal-motion never visits concave vertices of non-convex C-Obstacles.

The D*MECHA algorithm uses Proposition 5.1 and Proposition 5.2 in order to minimise the number of configurations considered for the construction of the RVG_g . Note that these propositions are applied in the two-dimensional projection of the CTP_i s on the x-y plane.

5.5.1 The D*MECHA Algorithm

The D*MECHA algorithm starts from configuration q_s at t_0 and it iteratively generates the RVG_g , in CT. It then searches this graph for the shortest Euclidean path q_s to q_g . This path corresponds to the time-minimal motion from q_s to q_g in C. The algorithm starts by expanding configuration q_s at time t_0 , it identifies all the q_s -visible and reachable configurations and then it rejects all the non-super-extremes of the q_s -visible sequences and all the concave vertices. It places the remaining visible configurations on a list called OPEN, which contains all the candidate configurations for expansion next. A function \hat{t}_f is evaluated for each configuration on OPEN and the configuration with the smallest \hat{t}_f is chosen for expansion next.

The evaluation function \hat{t}_f is defined such that its value $\hat{t}_f(q_n)$ at any configuration q_n is an estimation of $t_f(q_n)$. Where $t_f(q_n)$ is the time-cost of travelling along the path traced by the actual time-minimal motion from q_s to q_g constrained to pass through configuration q_n of vertex n. More formally,

$$\hat{t}_f(q_n) = \hat{t}_g(q_n) + \hat{t}_h(q_n) \quad (5.3)$$

$\hat{t}_g(q_n)$ is an estimation of the actual time-minimal motion from the q_s configuration to the current configuration q_n . The function $\hat{t}_h(q_n)$ is an estimation of the actual time-minimal motion from the current configuration q_n to q_g . An obvious choice for $\hat{t}_g(q_n)$ is the time-cost of the path from q_s to q_n the algorithm already found. The choice $\hat{t}_h(q_n)$ is not so obvious, therefore information about the problem domain should be taken into consideration. The airline distance between the current configuration q_n and the q_g half-line can be chosen as a heuristic estimate. Note that the distance to the q_g half-line should be measured from the current configuration q_n to a configuration q_g' on the q_g half-line in such a way, that the line-segment $\overline{q_n q_g'}$ creates an angle ϑ with the x-y plane. The travel time along the airline distance from q_n to the q_g at angle ϑ is the fastest possible travel time, so $\hat{t}_h(q_n)$ underestimates $t_h(q_n)$.

Every configuration produced is also marked visited and is placed in the search graph RVG_9 with a pointer to its parent node, thus generating a spanning tree called Motion of the RVG_9 . D*MECHA algorithm carries on in this manner iteratively until the q_g configuration is reached (this is when it is picked from the OPEN list for expansion) or when there are no further candidate configurations for expansion on OPEN. Note that if after the expansion of the configuration, say q_{exp} , a configuration, say $q_{neighb'}$, (which corresponds to the neighb vertex of an obstacle) is produced, then if this node has been visited before at configuration say q_{neighb} , and the new way of attaining it gives rise to a faster motion (time-wise) than that previously encountered, then the tree Motion is updated by redirecting q_{neighb} 's pointer towards q_{exp} , its \hat{t}_f value is updated and placed

on OPEN if its not there already, for reconsideration for expansion next. Note that in this case q_{neighb} is not the same with $q_{\text{neighb'}}$ but according to Lemma 5.1 even though an C-Obstacle's vertex can be visible and reachable in several configurations the one with the smallest reachable time is retained. For example, as can be noticed in Figure 5.9 the shaft edge 2 of the CTP_1 can be intersected by cones, which emanate from two different configurations and specifically from the q_s configuration and from configuration y . This happens because it is possible for the same vertex of a C-Obstacle to be visible and reachable in different time instances and locations in C from different vertices. This situation corresponds to different intersections of the sweeping half-line with the same shaft edge in different time instances from different configurations. However, configuration q_{neighb} is retained because it has the youngest reachable time. In this way at each iteration of the algorithm the tree Motion reflects the best path in CT explored so far.

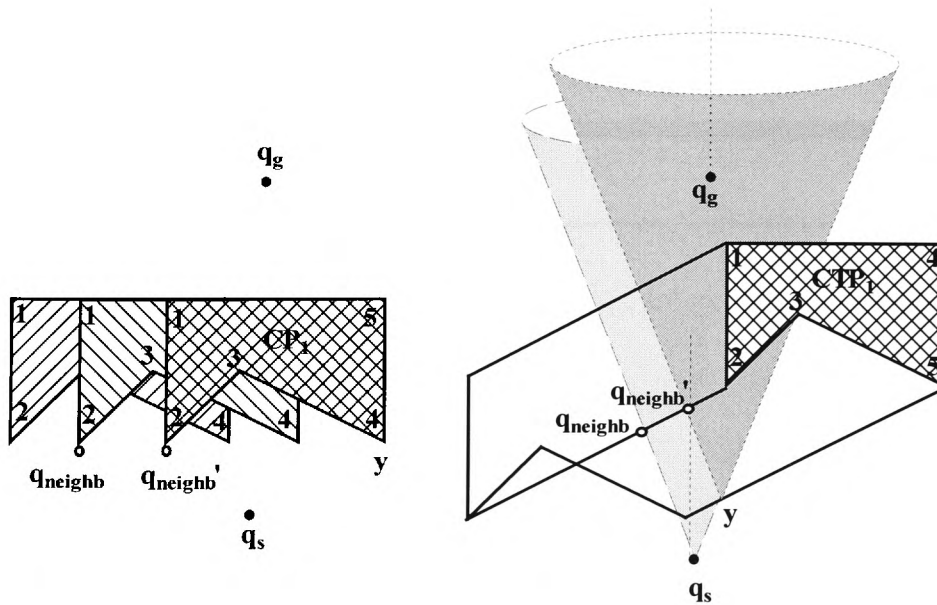


Figure 5.9 Treatment of the shaft edges considered for the construction of the RVG_9 that have been visited before.

The algorithm terminates either when it reaches the q_g configuration or when there are no more configurations on OPEN for expansion. In the first case the time-minimal semi-free motion from q_s to q_g is obtained by backtracking all the pointers in Motion from the q_g to q_s . In the second case there is not a path between q_s and q_g in CT, which means that there is not a semi-free motion from the AGV's start point to its goal point and the algorithm just reports failure.

Note below that the input of the algorithm is the CT configuration space. Since the obstacles shapes, positions and velocities are accurately known ahead of planning, the AGV's CT can be constructed easily. The proposed algorithm finds the time-minimal motion and is stated as follows.

D*MECHA Algorithm

INPUT:	AGV's q_s and q_g half lines and CTP _i obstacles.
OUTPUT:	Time-minimal collision-free motion from AGV's start point to its goal point.

```

begin
     $\theta := \arctan (v_{\max}^{-1})$ 
    put  $q_s$  in Motion;
    put  $q_s$  on OPEN;
    mark  $q_s$  visited;
     $\hat{t}_g(q_s) := 0$ ;
    while (OPEN  $\neq$  nil) do
        begin
             $w := \{q_i \in \text{OPEN} : \hat{t}_f(q_i) < \hat{t}_f(g_j) \mid \forall q_j \in \text{OPEN}, \text{ resolve ties arbitrarily but always in favour of the } q_g \text{ configuration}\}$ ;

```



```

remove w from OPEN;
if w = qg then exit while loop;
VV    := {VISIBLE_CONFIGURATIONSg(w, CTP)};
SE    := {SUPER_EXTREMES(w, VV)};
Mark all shaft edges, which correspond to {VV-SE} configurations
useless;
NCV   := SE - {concave vertices in SE};
for each vertex qi ∈ NCV do
  if qi is not marked useless then
    if qi is not marked visited then
      begin
         $\hat{t}_h(q_i) := t(d_{air}(q_i, q_g))_g$  (the time of travelling along the
        airline distance from qi to qg at angle g);
         $\hat{t}_g(q_i) := \hat{t}_g(w) + t(d_{edge}(w, q_i))$ ;
         $\hat{t}_f(q_i) := \hat{t}_g(q_i) + \hat{t}_h(q_i)$ ;
        put qi in Motion with pointer toward w;
        put qi in Open;
        mark qi visited;
      end;
    else if  $\hat{t}_g(q_i) > \hat{t}_g(w) + t(d_{edge}(w, q_i))$  then
      begin
        redirect pointer of qi toward w in Motion;
        if qi ∈ OPEN then remove i from OPEN;
         $\hat{t}_g(q_i) := \hat{t}_g(w) + \hat{t}_g(d_{edge}(w, q_i))$ ;
         $\hat{t}_f(q_i) := \hat{t}_g(q_i) + \hat{t}_h(q_i)$ ;
        put qi on OPEN;
      end;
  end;
end;
if w = qg then return the motion by tracing all the pointers in Motion from qg
back to qs else if OPEN = nil then return failure;
end.

```

In the above algorithm, Motion is a search tree, which represents at any instant the best motion obtained so far. For each visited configuration n (except q_s) a pointer to its parent is held. The function $\hat{t}_r(q_n)$ is associated with each configuration n in the current Motion. The $t(d_{\text{edge}}(w, q_n))$ is a function, which represents the time-cost of travelling (with v_{\max_R}) along an edge, which connects two configurations in RVG_θ . The $t(d_{\text{air}}(w, q_n))_\theta$ is a function, which represents the time-cost of travelling, with v_{\max_R} along the airline distance between configurations w and q_n at angle θ . The list OPEN contains at any instant, the configurations that are candidates for consideration next. All the configurations of the environment are initially marked as unvisited and useful.

5.5.2 The D*MECHA's Subroutines

The first subroutine used by the D*MECHA algorithm is the ***VISIBLE_CONFIGURATIONS_{\theta}***. This subroutine is similar to the one presented in section 4.6.2 with some minor changes. It takes as arguments a configuration q and a set of CTP-Obstacles in CT and returns a set W of all the reachable and visible configurations from configuration q . The second subroutine is the ***SUPER_EXTREMES***. This routine is exactly the same as the one in section 4.6.2 therefore will be not presented here again. The ***SUPER_EXTREMES*** routine takes as arguments a configuration q and a set W of reachable and visible configurations from q and returns the super-extremes of the q -visible sequences.

VISIBLE_CONFIGURATIONS_{\theta}(q_v, CTP)

INPUT: A collision-free configuration q_v and the set CTP.

OUTPUT: The set W of all vertices visible from vertex v .

begin

Sort the CT's shaft edges according to the clockwise angle the projection to the x-y plane of the half-line emanating from q_v at angle θ with the x-y plane, through each shaft edge, create with the x-axis. If there are any ties give priority according to their distance to q_v . Let q_1, q_2, \dots, q_n be the sorted list;

Let ℓ be the half-line emanating from q_v at angle θ with the x-y plane, the projection of which to the x-y plane is parallel to the x-axis. Find the CTP_i's faces that are properly intersected by ℓ (intersected in other configurations than the CTP_i's shaft edges) and store them in a balanced tree T in the order their intersection;

$W := \emptyset$;

for $i := 1$ **to** n **do**

if **VISIBLE** (q_i) **then add** q_i **to** W ;

Insert into T the CTP_i's face incident to q_i that lie on the clockwise side of the half-line from q_v to q_i ;

Delete from T the CTP_i's face incident to q_i that lie on the counter-clockwise side of the half-line from q_v to q_i ;

return W ;

end.

The **VISIBLE** subroutine called by **VISIBLE_CONFIGURATIONS_g** subroutine and decides whether configuration q_i is visible from q_v .

VISIBLE (q_i)

begin

if $\overline{q_v q_i}$ intersects the interior of the CT-Obstacle of which q_i is a shaft edge, locally at q_i **then return false**

else if $i=1$ **or** q_{i-1} is not on the segment $\overline{q_v q_i}$

then Search in T for the face f in the leftmost leaf

```

        if f exists and  $\overline{q_v q_i}$  intersects f
            then return false
        else return true
    else if  $q_{i-1}$  is not visible
        then return false
    else Search in T for a face f that intersects  $\overline{q_{i-1} q_i}$ 
        if f exists
            then return false
        else return true
end.

```

The computational complexity of the *VISIBLE_CONFIGURATIONS*_g subroutine is $O(n \log n)$, where n is the total number of the C-Obstacles' vertices. Indeed, as it can be noticed the first step of the algorithm, which sorts the configurations according to the clockwise angle requires $O(n \log n)$ time, where n is the total number of the C-Obstacles' vertices. The second step, checks for possible intersections between the half-line ℓ and the faces of the CTP_s and stores these intersections in the balance tree T . This step requires $O(n \log n)$ time, where n is the total number of the CT-Obstacles' shaft edges. The final step is the **for** loop, which is traversed n times. Each time the subroutine *VISIBLE* is called, which requires constant computational time for some geometric checks and $O(\log n)$ time for a constant number of operations on the balanced tree T , where n is the total number of the C-Obstacles' vertices. Therefore the computational complexity of the final step is $O(n \log n)$, where n is the total number of the C-Obstacles' vertices.

The discussion of the D*MECHA's subroutines completes the presentation of the algorithm. In section 5.6 a simple motion planning problem will be solved using the D*MECHA algorithm in order to test the algorithm's capabilities. This problem is taken from (Fujimura, 1991).

5.6 A Simple Motion Planning Problem

Consider the environment of Figure 5.10 populated by two moving obstacles P_1 and P_2 as well as the AGV's start point s and its goal point g .

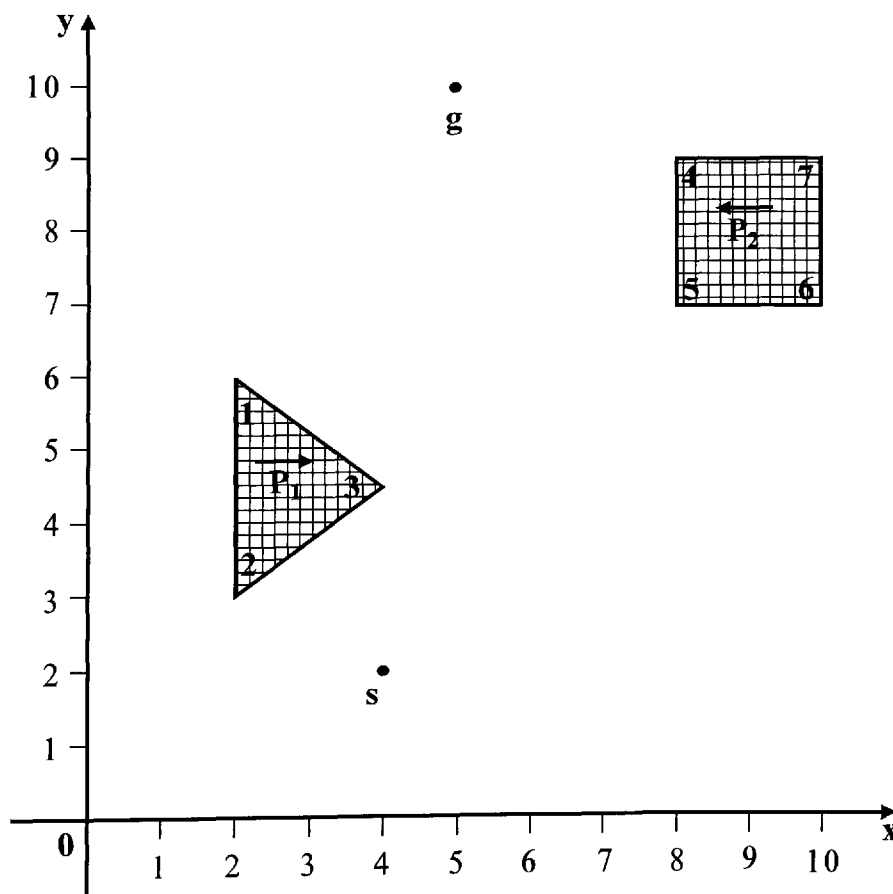


Figure 5.10 Illustration of the problem's scene.

The obstacles are moving in the directions indicate. Suppose that the velocity of both obstacles is 1m/sec and the maximum velocity that the robot can reach is 2m/sec. The obstacles start moving towards the indicated direction at the same time $t = 0$ and their motion terminates after 6 seconds. The positions of the AGV's start and goal locations and the obstacles initial locations as well as their shape are accurately known ahead of planning and they are as depicted in Figure 5.10. The co-ordinates of the AGV's start and goal points and obstacle's vertices at their initial locations, are given in Table 5.1.

Points	Co-ordinates
s	(4, 2)
g	(5, 10)
1	(2, 6)
2	(2, 3)
3	(4, 4.5)
4	(8, 9)
5	(8, 7)
6	(10, 7)
7	(10, 9)

Table 5.1 Co-ordinates of the AGV's start and goal locations and the obstacles' vertices initial locations of the environment of Figure 5.10.

Taking into consideration the positions of the AGV's start point s its goal point g and the obstacles' initial positions and their velocity, the AGV's space-time configuration space CT can be constructed. The AGV's space-time configuration space CT is as depicted in Figure 5.11.

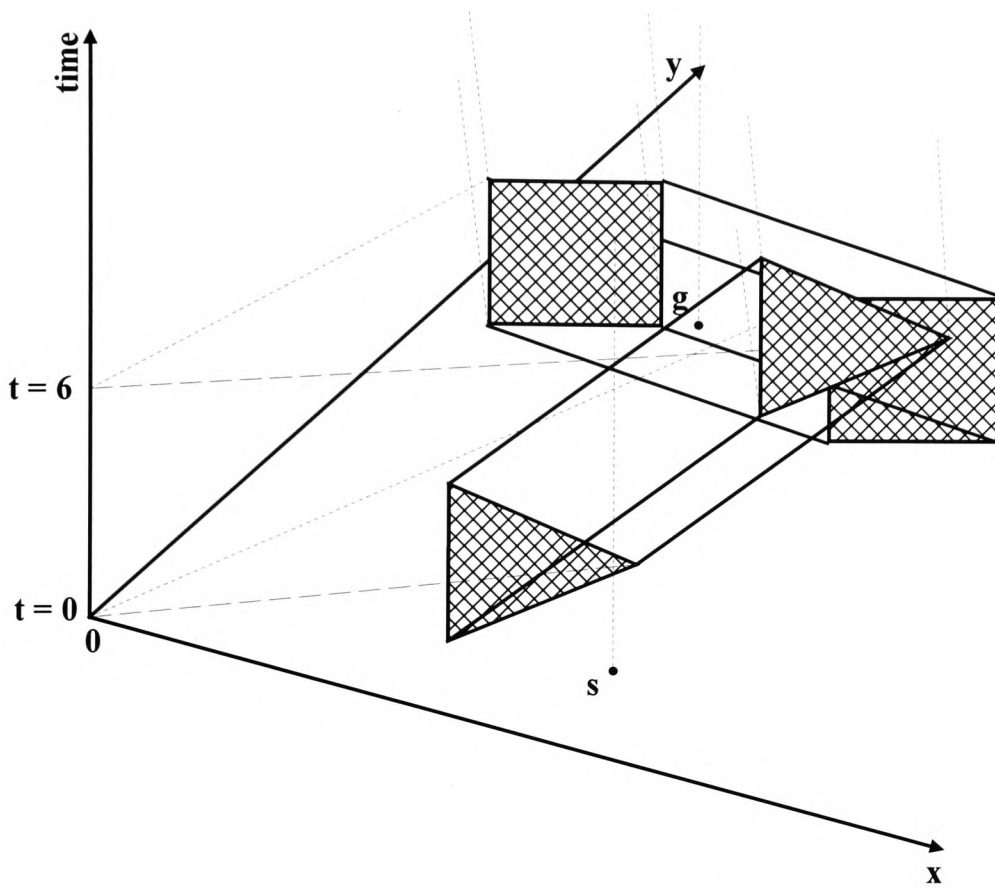
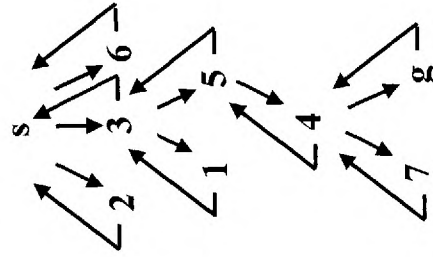


Figure 5.11 Illustration of the AGV's space-time configuration space CT.

To test the D*MECHA, the CT is fed to it as input and the values of its variables at each iteration along with its outputs are presented in Table 5.2. In Figures 5.12a-d the configurations, which are considered for the construction of the RVG_g at any iteration of the algorithm are depicted.

In table 5.2 and in Figures 5.12a-d the configurations on the CT-Obstacles' shaft edges will not carry the prefix q . The change in symbolisation is taking place purely for space matters.

Iterations	i	OPEN	w	VV	SE	NCV	useless	Visited	$\hat{t}_g(i)$	$\hat{t}_u(i)$	$\hat{t}_t(i)$
1	s	s						s			
	2	2	s	2,3,6	2,3,6	2,3,6		s ₂	0.75	3.62	4.37
	3	2,3						s _{2,3}	1.5	2.75	4.25
	6	2,3,6						s _{2,3,6}	3	1.75	4.75
2	s	2,6	3	s,1,2,5,6	s,1,2,5,6	s,1,2,5,6					
	1	1,2,6						s _{2,3,6}	? 0 > 1.5+1.5		
	2	1,2,6						s _{1,2,3,6}	2.4	2.0	4.4
	5	1,2,5,6							? 0.75 > 1.5+0.9		
	6							s _{1,2,3,5,6}	2.75	1.5	4.25
3	s	1,2,6	5	s,1,3,4,6	s,1,3,4,6	s,1,3,4,6			? 3 > 1.5+1.5		
	1								? 0 > 2.75+2.5		
	3								? 2.4 > 2.75+0.5		
	4	1,2,4,6							? 1.5 > 2.75+2.25		
	6							s _{1,2,3,4,5,6}	3.75	0.5	4.25
4	s	1,2,6	4	5,7,g					? 3 > 2.75+0.66		
	5										
	7	1,2,6,7						s _{1,2,3,4,5,6,7}	? 2.75 > 3.75+1.5		
	g	1,2,6,7,g							4.41	0.5	4.91
5	s	1,2,6,7	g					s _{1,2,3,4,5,6,7,g}	4.25	0	4.25

Motion**Table 5.2** Summary of the results of the D*MECHA algorithm when is applied to the problem of Figure 5.10.

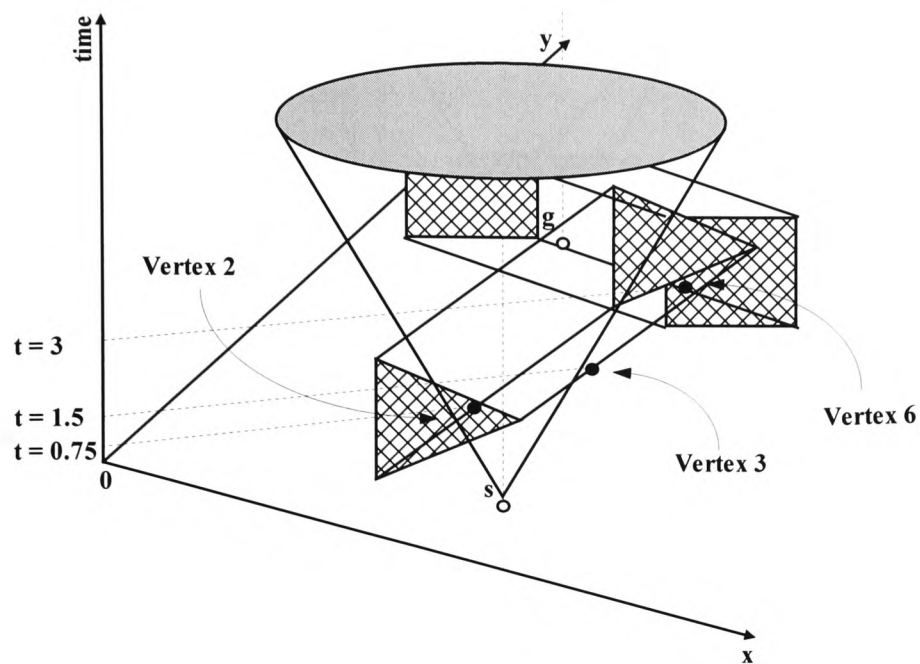


Figure 5.12a The black dots indicate all the configurations placed in the tree Motion at the first iteration of the algorithm.

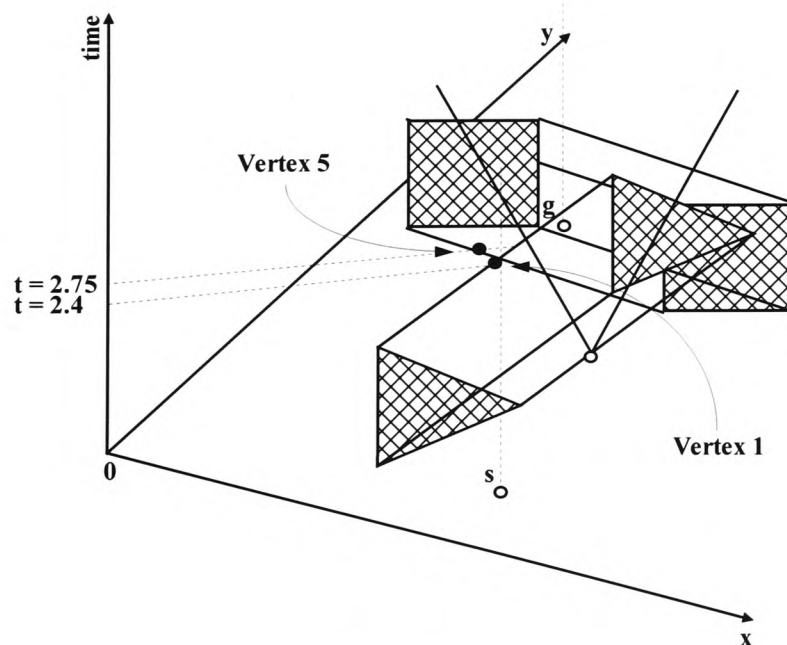


Figure 5.12b The black dots indicate all the configurations placed in the tree Motion at the second iteration of the algorithm

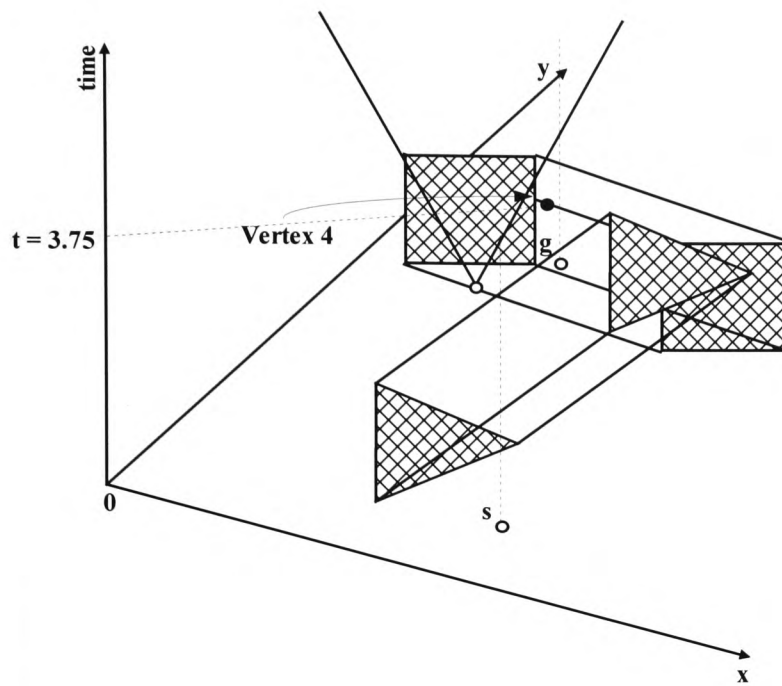


Figure 5.12c The black dots indicate all the configurations placed in the tree Motion at the third iteration of the algorithm.

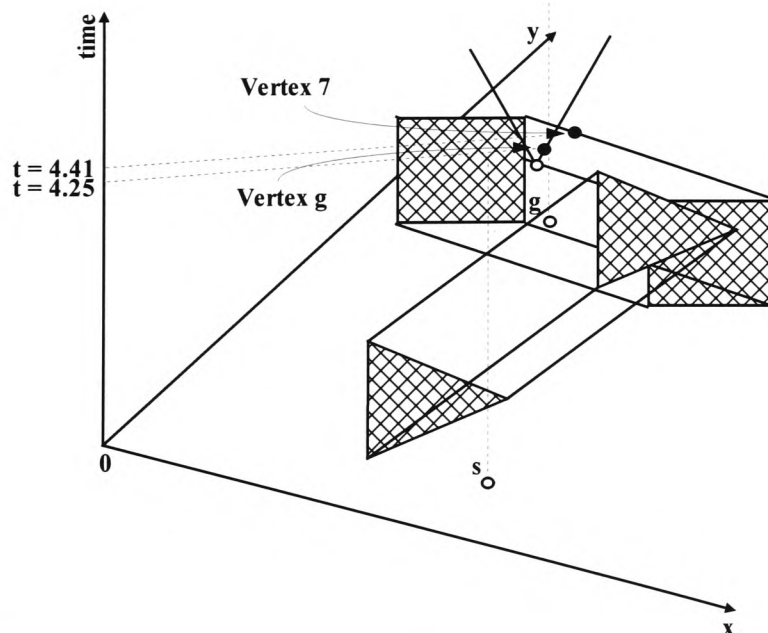


Figure 5.12d The black dots indicate all the configurations placed in the tree Motion at the fourth iteration of the algorithm.

After the D*MECHA algorithm terminates with $w = t_g$, the time-minimal semi-free motion for the AGV between its start and goal points can be obtained by backtracking all the pointers in the search tree Motion from t_g back to t_s . Therefore for the problem of figure 5.10, the time-minimal semi-free motion from start to goal is $\text{Motion} = \{t_s, t_3, t_5, t_4, t_g\}$ and its time-cost is 4.25 seconds. Figure 5.13 depicts the motion produced by the D*MECHA algorithm for the environment of Figure 5.10.

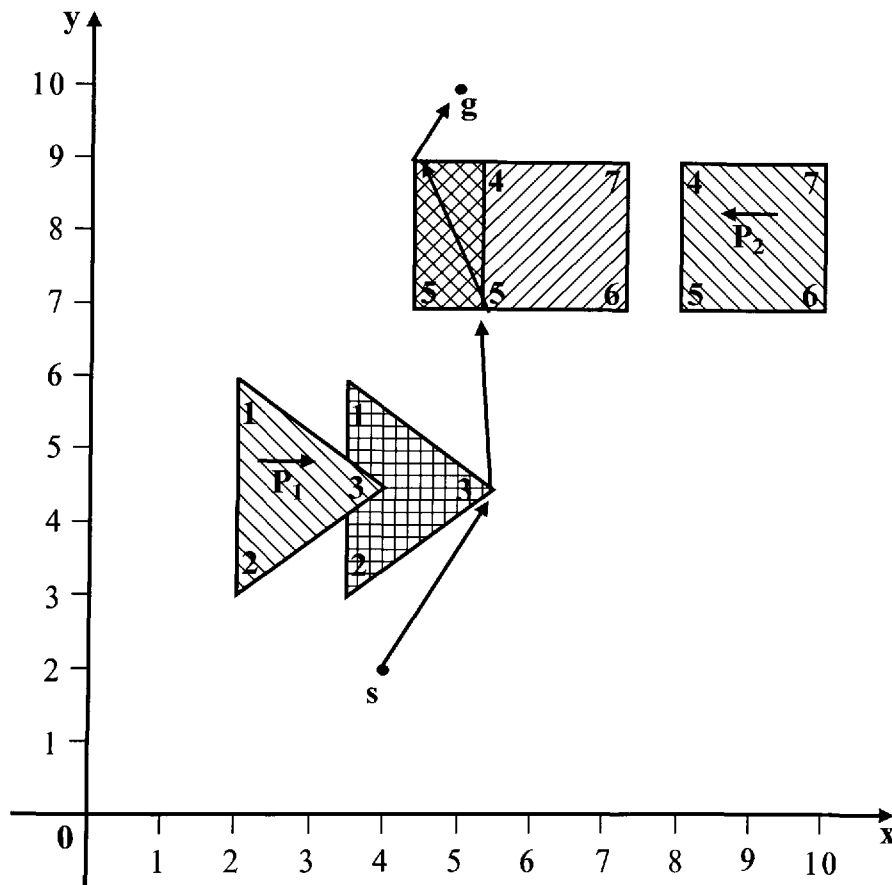


Figure 5.13 The arrows illustrate the semi-free motion proposed as a solution to the problem of Figure 5.10 from the D*MECHA algorithm.

This motion is a semi-free and is also the time-minimal motion between start and goal as it will be shown in section 5.8. Note that while the AGV moves from vertex 5 to vertex 4 between times $t_5 = 2.75$ min and $t_4 = 3.75$ min, it might coincide with some configurations on the edge (4, 5).

5.7 The Admissibility and the Optimality of the D*MECHA Algorithm

As was mentioned in section 5.1 the D*MECHA algorithm is based on the V*MECHA algorithm presented in chapter four. The proofs of the D*MECHA's admissibility and optimality are the same as the V*MECHA's proofs presented in section 4.8 and section 4.9 respectively. Therefore these proofs are omitted from this chapter.

The D*MECHA algorithm is admissible, which means that it always produces the time-minimal motion from q_s to q_g within the RVG_9 (providing that one exists), otherwise it returns failure. Also the D*MECHA algorithm is optimal in the sense that it never expands more configurations than any other admissible algorithm, which is less or equally informed, for the identification of the time-minimal motion.

5.8 The Optimality of the Motion Produced by the D*MECHA Algorithm

According to Theorem 5.1, the time-minimal semi-free motion for an AGV in an environment populated by moving and stationary obstacles from its start point to its goal point, turns only at obstacles' vertices. Therefore the visibility graph at angle θ in CT encapsulates the time-minimal semi-free motion. Proposition 5.1 and Proposition 5.2 ensure that the RVG_θ encapsulates the time-minimal semi-free motion. Therefore, since the D*MECHA algorithm is admissible it guarantees that it finds the time-minimal motion within the RVG_θ . In order for the motion established by the D*MECHA algorithm to be the global time-minimal semi-free motion, it must be shown that there is no *stop-motion*, which can be accomplished by the AGV in less time than the one suggested by the D*MECHA algorithm. Stop-motion is the motion which involves stops and it can be defined in such a way, that the AGV stops at certain locations waits there until one or more obstacles move out of its way and then starts moving again towards the goal point.

Lemma 5.3

The time-minimal semi-free motion for an AGV R from its start location to its goal location in an environment populated by only one static or moving obstacle P_1 , is a non-stop-motion.

Proof

The proof of the Lemma is carried out in the two-dimensional configuration space C. For the case when CP_1 is static the Lemma follows immediately. In the case of a

moving obstacle, consider its configuration space in Figure 5.14. q_s and q_g are the AGV's start and goal configurations respectively and CP_1 is a moving C-Obstacle at its initial position. The C-Obstacle starts moving at time t_0 towards the direction indicated.

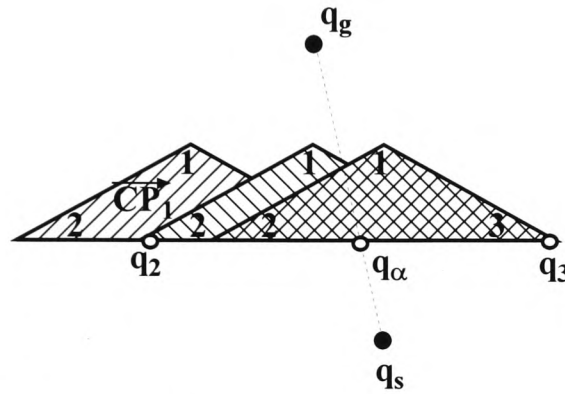


Figure 5.14 Illustration of the case where q_g is visible from q_2 at time t_2 .

The least time consuming motion for the AGV from q_s to q_g is the motion along the straight line-segment, which connects them, while R is moving with its maximum velocity (\bar{v}_{\max_R}). However, if the AGV starts moving at time t_0 , from q_s to q_g along this motion it will eventually collide with the CP_1 at configuration q_α at time t_α . Therefore R will have to circumnavigate CP_1 from either side. R can meet vertices 2 and 3 at configurations q_2 and q_3 at times t_2 and t_3 respectively, with $t_2, t_3 > t_\alpha$. Without loss of the generality suppose that the motion from the left hand side of the CP_1 is less time consuming than the one from the right hand side of it. As mentioned earlier R can meet vertex 2 at configuration q_2 at time t_2 . When the AGV is at configuration q_2 at time t_2 , either the q_g is visible or not visible. If the q_g is visible (Figure 5.14), then the AGV can start moving towards it at time t_2 and reach it.

According to the formulation of the problem the velocity of any obstacle in the environment is less than the AGV's velocity. Therefore, by using triangle inequalities over $\{q_2, q_\alpha, q_g\}$, it can be obtained that motion $\{q_2, q_g\}$ is faster than motion $\{q_2, q_\alpha, q_g\}$. Thus $\{q_s, q_2, q_g\}$ is faster than motion $\{q_s, q_2, q_\alpha, q_g\}$ and so is faster than the stop-motion $\{q_s, q_\alpha, q_g\}$, hence it is time-minimal. Also note that if the AGV starts moving from q_2 to q_α at time t_2 it will get to q_α in less time than the CP_1 's vertex 2 will, because its velocity is larger than that of the CP_1 , this fact further endorses the triangle inequality. The minimality of the motion $\{q_2, q_g\}$ over $\{q_2, q_\alpha, q_g\}$ also follows immediately by Lemma 5.2. In the same manner time-minimality holds for one-dimensional obstacles.

If q_g is not visible from q_2 at the time instance t_2 then there are two possible outcomes. The first is that q_g will be visible from q_2 , while the AGV is moving from q_2 to q_g with constant velocity equal to \vec{v}_{\max_R} and the CP_1 is moving in the direction indicated. Note that the visibility is to be identified in CT. The visibility depends on the angle ϑ of the sweep line and the slope of the CTP_1 in CT, this means that the visibility depends on the velocities of the AGV and the C-Obstacle respectively. Figure 5.15 illustrates such a case.

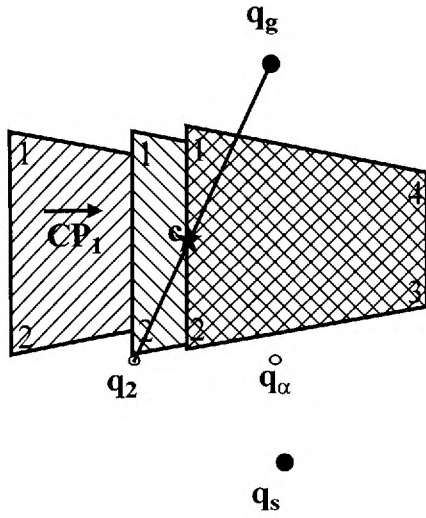


Figure 5.16a Illustration of the case where q_g is not visible from q_2 at t_2 and it does not become so even while the AGV is moving towards q_g .

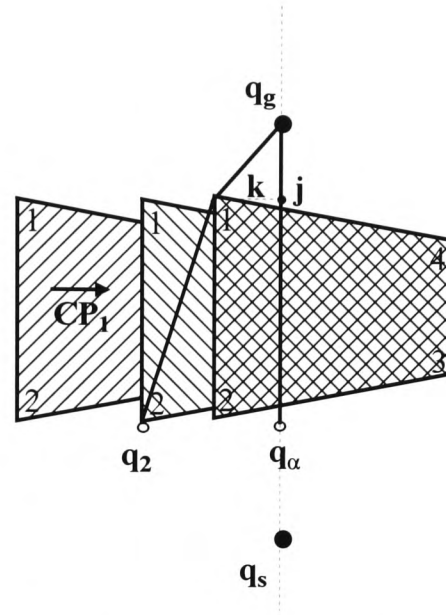


Figure 5.16b For the case depicted in Figure 5.16a the AGV will have to pass through another vertex of the CP_1 on its way to q_g .

Figure 5.16a illustrates the case, where if the AGV starts moving from q_2 at time t_2 towards the q_g , with velocity equal to \bar{v}_{\max_R} , it will eventually collide with the CP_1 at the point c . Therefore the AGV will have to pass through another vertex of the CP_1 before reaching q_g . Figure 5.16b illustrates the motion of the AGV, in which it will have to pass through the CP_1 's vertex 1. Note that the visibility is to be identified in CT.

For the proof of the lemma what remains to be shown is that the motion $\{q_s, q_2, q_1, q_g\}$ is less time consuming than the stop-motion $\{q_s, q_\alpha, q_g\}$. Since the AGV at time t_2 can be either at configurations q_2 or q_α it has to be shown that the motion from q_2

to q_g through q_1 is faster than the motion from q_2 to q_g through q_α . If the AGV leaves q_2 at time t_2 it will reach configuration q_1 at time t_1 at a configuration on the half plane defined by a line passing through q_α and q_g and contains q_2 . Otherwise q_g would have been visible from q_2 while both the AGV and the CP_1 were in motion. The fact that vertex 1 at t_1 is on the half-plane which contains q_2 means that q_g is not visible from q_α at t_1 . So if the AGV departs from q_1 at t_1 and moves along a motion, which is parallel and in the same direction as the direction of the CP_1 's motion, it will eventually join the stop-motion $\{q_s, q_\alpha, q_g\}$ at a configuration j and time say t_j . By using triangle inequalities over $\{q_1, j, q_g\}$ it is derived that the motion $\{q_1, q_g\}$ is less time consuming than the motion $\{q_1, j, q_g\}$. But since t_j is the very latest time that the AGV could reach j in the best case scenario depending on the shape of the C-Obstacle (Figure 5.16b does not illustrates the best case scenario) by following the stop-motion $\{q_s, q_\alpha, q_g\}$, it is derived that the motion $\{q_s, q_2, q_1, q_g\}$ is less time consuming than the motion $\{q_s, q_2, q_\alpha, q_g\}$. Therefore motion $\{q_s, q_2, q_1, q_g\}$ is less time consuming than the stop-motion $\{q_s, q_\alpha, q_g\}$ and hence is time-minimal. Note that while the AGV follows the motion $\{q_s, q_2, q_1, q_g\}$, it might coincide with the boundary of CP_1 for some time. Hence, the Lemma holds.

Theorem 5.2 (The Time-Minimal Motion Theorem)

The time-minimal motion for an AGV R from its start location s to its goal location g in an environment populated by a set P of static and moving obstacles, is a non-stop-motion.

Proof

The proof of the Theorem is carried out in the two-dimensional configuration space C . The theorem is proved by induction on n , the total number of C-Obstacles in C . In the base case this is when $n = 1$ (one C-Obstacle in the C) the theorem holds by Lemma 5.3.

Assume that the theorem holds for $n-1$ C-Obstacles in the environment. The induction hypothesis states that if the AGV departs at any time from q_s , a time-minimal collision semi-free motion μ exists from q_s to q_g in an environment populated with $n-1$ C-Obstacles and this is a non-stop-motion. Thus if there are $n-1$ C-Obstacles in the environment and the AGV departs from q_s at time, say t_s , a collision-free non-stop motion μ exists, which if the AGV follows it will arrive at q_g at time, say t_g . This means that if the AGV departs from the q_g at time t_g , and the C-Obstacles have reverse motion it can be at q_s at time t_s by following the μ motion reversed.

The last step of the proof is to show that the theorem holds for n C-Obstacles. Assume that if the environment contains n C-Obstacles and the AGV departs from q_s at time t_s it can be at q_g at time, say t_g' by following a time-minimal motion, it is not yet known whether this motion is a stop-motion or a non-stop-motion.

Suppose now that in the environment with $n-1$ C-Obstacles, a C-Obstacle is inserted so the total number of C-Obstacles is n . If the AGV does not collide with the newly inserted C-Obstacle, while is moving along motion μ then the theorem holds. In this

case the AGV departs from q_s at time t_s and gets to q_g at time $t_g' = t_g$. If the AGV collides with the newly inserted C-Obstacle, while it is moving along motion μ , then $t_g' \neq t_g$.

Suppose further that the last C-Obstacle's vertex that the AGV passes via, before the collision with the newly inserted C-Obstacle occurs, is vertex v at configuration q_v at time t_v . By the induction hypothesis it is known that the motion from s to v is time-minimal and a non-stop motion. It is also known by Lemma 5.3 that if the AGV departs from q_v at time t_v it will avoid collision with the newly inserted C-Obstacle by realising a time-minimal, non-stop motion to some vertex of an C-Obstacle that has not been considered yet for the construction of the time-minimal motion or to the q_g' . Let this vertex be vertex w . If $w = q_g'$ then this motion is time-minimal and non-stop and $t_w = t_g'$. If w is a vertex of a C-Obstacle then the motion from w to q_g is a time-minimal non-stop motion (recall that if the AGV departs from q_g' at time t_g' and the C-Obstacles have reverse motion it can be at q_s at time t_s' by following the motion reversed). Therefore time-minimal motion from s to g in an environment populated by a set P of n static and moving obstacles is a non-stop-motion. And the theorem holds.

Corollary 5.1

The motion established by the D*MECHA algorithm for an AGV which only translates between two query points, in a two-dimensional environment populated moving and static simple polygonal obstacles, is the global time-minimal motion.

Proof

The proof follows immediately from Theorem 5.1, Proposition 5.1, Proposition 5.2 and Theorem 5.2.

5.9 Time and Space Complexities of the D*MECHA Algorithm

In this section an empirical analysis of the computational time and space of the D*MECHA algorithm will be presented.

Theorem 5.3

The D*MECHA algorithm establishes the time-minimal semi-free motion for an AGV, between two query points in an environment populated by both static and moving obstacles, in $O(k n \log n)$ computational time and in $O(k^2)$ space, where n is the total number of the obstacles' vertices and k is the total number of the obstacles' non-concave vertices.

Proof

The length of the list OPEN is $O(k)$, where k is the total number of the C-Obstacles' non-concave vertices, thus the **while** loop of the algorithm is traversed at most k times. Therefore there are $O(k)$ iterations and at each iteration the following steps are executed. The identification of the configuration on OPEN with the smallest $\hat{t}(q_i)$ value requires $O(k)$ time, where k is the total number of the C-Obstacles' non-concave vertices. The identification of all the w-visible and reachable configurations at angle θ requires $O(n \log n)$ time, where n is the total number of the C-Obstacles'

vertices, see section 5.5.2 for details. The identification of the extreme configurations of any w -visible sequence requires $O(n \log n)$ time, where n is the total number of the C-Obstacles' vertices, see section 5.5.2 for details. The treatment of w 's children requires $O(k)$ time, where k is the total number of the C-Obstacles' non-concave vertices.

The overall computational time of the D*MECHA algorithm is in $O(k n \log n)$, where n is the total number of the C-Obstacles' vertices and k is the total number of the C-Obstacles' non-concave vertices.

The number of edges of the RVG_9 produced by the D*MECHA algorithm is bounded by $\binom{k+2}{2}$. Since $\binom{k+2}{2} = \frac{(k+2)(k+1)}{2}$ which is in $O(k^2)$, the space complexity of the D*MECHA algorithm is in $O(k^2)$.

The worst time and space complexities are attained by the D*MECHA algorithm when it is applied in an environment populated by only convex obstacles and k becomes equal to n leading to a computational time $O(n^2 \log n)$ and $O(n^2)$ space.

5.10 Some Interesting Properties of the Motion's Time-Minimality

As was seen from the above discussions the D*MECHA algorithm establishes the time-minimal semi-free motion between two query points for an AGV in an environment populated by moving obstacles, in $O(k n \log n)$ computational time and

(k^2) computational space, where k is the total number of the obstacles' concave vertices and n is the total number of the obstacles' vertices. The time-minimality characteristic of the produced motion demonstrates some very interesting properties. These properties enable the algorithm to provide solutions to other related problems. For instance the algorithm can provide a solution to the decision problem of whether a semi-free motion between the AGV's start point and its goal point that is accomplished by a given time (deadline) exists.

Another problem that can be solved by the D*MECHA algorithm is the one which requires the establishment of the latest departure time for the AGV from its start point in order to arrive at its destination point at a given time. This property is very important when the AGV's motion is part of a manufacturing process in which there are different tasks that have to be co-ordinated. For the solution to this problem the goal point and the arrival time are set as start point and start time of the AGV respectively and the obstacles have reverse motions. Notice that now the motion finding process is reversed. The time that the AGV arrives at the destination point (the original start point) by following the time-minimal motion is the solution to the problem.

Notice that when there are only stationary obstacles in the environment, the time-minimal motion from s to g defines also the shortest path from s to g .

Note also that some of the assumptions made in section 5.1 can be relaxed but as was mentioned in section 5.1, unfortunately at the expense either of the computational

time of the algorithm or of the minimality of the produced motion or even at the expense of the solution's completeness. For instance, the assumption that the AGV always moves always with greater velocity than the obstacles. If there is an obstacle in the environment that moves faster than the AGV the algorithm may fail to find the time-minimal motion because it may pay for the AGV to follow a stop-motion.

5.11 Comparing the D*MECHA Algorithm to other Approaches

In this section the D*MECHA algorithm is compared with other approaches that have been proposed over the years for the solution of the robot motion planning problem.

As was mentioned in section 5.2 Kant and Zucker (1986), decompose the trajectory planning problem (TPP) into two sub-problems. (i) The path planning problem (PPP) in which a path, which avoids collisions with static obstacles, is planned and (ii) the velocity planning problem (VPP) in which the velocity, which avoids collisions with moving obstacles along this path, is planned. Their approach is not complete and it might fail to produce a motion when one exists. The reason for this is that the AGV is not allowed to circumnavigate the obstacles but only move along the path established at the path planning stage of the approach. Therefore if an obstacle moves along the AGV's path or its motion terminates on the AGV's path then the approach fails to find motion even though one exists. The advantage of the D*MECHA algorithm over this approach is that it always establishes a path if one exists (under the assumptions made in section 5.1). Therefore it succeeds to produce

a semi-free motion even in situations where Kant and Zucker's approach fails. Another major advantage of the D*MECHA algorithm over the approach of Kant and Zucker is that the motion that it produces is time-minimal while Kant and Zucker's approach is far from time-minimal because it involves many stops.

Erdmann and Lozano-Pérez (1987), presented the space-time configuration space as a list of configuration space slices at particular points in time. These points in time are those at which a moving object changes its velocity. The motion of the robot consists of a series of straight-line segments connecting nodes of different slices and the robot moves with constant velocity between two slices. Along this motion the robot changes its velocity only at nodes of obstacles when an obstacle's velocity changes. Their algorithm is time resolution-complete between the slices. The algorithm runs in time $O(r n^3)$, where n is the total number of the environment's edges and r is the total number of the constructed slices. The advantage of the D*MECHA algorithm over this algorithm is that it is computationally more efficient. It establishes the time-minimal semi-free motion in $O(k n \log n)$ time, where k is the total number of the obstacles' non-concave vertices and n is the total number of the obstacles vertices. Also note that the D*MECHA algorithm considers far fewer obstacles' vertices for the construction of the reacha-visibility graph in CT, which makes the search process for the time-minimal motion quicker.

Fujimura (1991) presented an algorithm to find a motion for a point-robot, in an environment populated by time-dependent obstacles and a destination point. The environment's obstacles are polygons, which move in a fixed direction at constant

speed. The algorithm they proposed finds the time-minimal motion given that the point-robot moves faster than the obstacles and the destination point. This algorithm is based on the concept of the accessibility graph. The algorithm searches this graph and finds the time-minimal motion from the robot's start point to its goal point in $O(n^2 \log n)$ computational time, n is the total number of the obstacles' vertices. The advantage of the D*MECHA algorithm over this algorithm is that it is computationally more efficient and that in general considers less obstacles' vertices for the construction of the reacha-visibility graph and thus makes the search process for the time-minimal motion less time consuming.

5.12 Discussion

In this chapter an algorithm for solving the dynamic robot motion planning problem was proposed. The algorithm is called D*MECHA and establishes the time-minimal semi-free motion for an AGV (point robot) between two query points in an environment populated by simple polygonal obstacles.

The computational complexity of the algorithm is $O(k n \log n)$, where k is the total number of the obstacles' non-concave vertices and n is the total number of the obstacles' vertices. The D*MECHA algorithm is an extension of the V*MECHA algorithm presented in chapter four therefore the two propositions used in order to minimise the number of vertices considered for the construction of the visibility graph were used here for the reduction of the visibility graph in CT without sacrificing the optimality of the motion.

The algorithm was proven to be admissible and optimal in the sense that it never expands more vertices than any other less or equally informed admissible algorithm. The time-minimality theorem was stated proving that the motion produced by the algorithm is time-minimal under some mild assumptions.

By the time-minimality theorem it was stated that in order for the AGV to attain a time-minimal semi-free motion should follow a vertex-to-vertex non-stop motion by moving constantly at its maximum velocity. However this may appear counter-intuitive to a human being. The reason for that is that the humans and in general every animal possess the cognitive ability to follow the quick and at the same time safe motion rather than a dangerous fast motion within a physical environment. This human ability is extremely hard to replicate in computer algorithms. Another factor is that the AGV is supposed to be a point subject only to velocity bounds while the real physical AGVs are dimensioned objects subject to an acceleration bound and other physical constrains such as friction, inertia and so on. Finally the assumption that the robot moves with greater velocity than the obstacles is not always realizable in physical environments. Taking into consideration all these factors a different strategy may have to be employed for the solution of the robot motion planning problem. In the next chapter the extensibility of the D*MECHA algorithm in more complicated environments will be investigated.

5.13 References

- CANNY, J. and REIF, J. 1987. New Lower Bound Techniques for Robot Motion Planning Problems. *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science*, Los Angeles, pp. 49-60.
- ERDMANN, M. and LOZANO-PÉREZ, T. 1987. On multiple moving objects. *Algorithmica*, **2** (4), pp. 477-522.
- FUJIMURA, K. 1991. *Motion Planning in Dynamic Environments*. Tokyo: Springer – Verlag.
- FUJIMURA, K. 1993. Motion planning in time-varying domains: the case of cyclic motions. *Advanced Robotics*, **7** (5), pp. 491-505.
- FUJIMURA, K., 1994. Motion Planning Amid Transient Obstacles. *The International Journal of Robotics Research*, **13** (5), pp. 395 - 407.
- FUJIMURA, K. and SAMET, H. 1993. Planning a Time-Minimal Motion Among Moving Obstacles. *Algorithmica*, **10**, pp. 41-63.
- KANT, K. and ZUCKER, S. W. 1986. Toward Efficient Trajectory Planning: The Path-Velocity Decomposition. *The International Journal of Robotics Research*, **5** (3), pp 72-89.

REIF, J. and SHARIR, M. 1985. Motion Planning in the Presence of Moving Obstacles. Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science, Portland, pp. 144-154.

SUTNER, M. and MAASS, W. 1988. Motion planning among time dependent obstacles. *Acta Informatica*, **26**, pp. 93-122.

6

Extensions to the D*MECHA Algorithm

I know that thou canst do everything

JOB 42: 2

6.1 Introduction

In chapter five the D*MECHA algorithm for finding the time-minimal motion between two query points in an environment populated by linearly moving obstacles was described. In this chapter the applicability of the D*MECHA algorithm in more complicated dynamic environments will be investigated. In particular two types of environments will be considered. The first contains obstacles, which change their size over time and the second contains obstacles, which have piecewise linear motion.

The extensibility of the D*MECHA algorithm to solve the dynamic motion planning problem for environments, which contain obstacles whose size changes over the time is presented. This means that the obstacles in the environment can shrink or expand over time as well as change their position at the same time. This is a slightly modified problem to the classic dynamic robot motion planning problem. In the classic problem the position of the obstacles in the environment is time dependent, whereas in this instance the size of the obstacles as well as their position in the environment depend on time. There are a large number of significant applications, which can be modelled using the concept of shrinking and expanding obstacles. In section 6.2 examples of such applications and the motivation behind this work will be discussed.

The extensibility of the D*MECHA algorithm to solve the dynamic motion planning problem for environments, which contain obstacles with piecewise linear motion is also investigated in this chapter. Most of the time a real world environment where an AGV operates, cannot be described adequately when only linearly moving obstacles are used for its formulation. Obstacles with piecewise linear motion, can describe more general environments in the sense that the motion of the obstacles is more relaxed and therefore the domain of problems that can be formulated using piecewise linearly moving obstacles is larger. An obstacle has piecewise linear motion when it moves for a finite number of time intervals in a fixed direction with constant velocity. Note that the obstacle's velocity in different time intervals does not have to be the same.

It will be shown that the D*MECHA algorithm with minor changes is still applicable to such environments. It will also be shown that the admissibility and the optimality

properties of the algorithm are maintained. The motion produced by the algorithm in such environments also maintains the property of its optimality. More formally the two problems considered in this chapter are as follows.

Problem 3 (Shrinking and Expanding Obstacles)

Consider the problem of planning the motion of an AGV R in a two-dimensional workspace W populated by simple polygonal convex obstacles P_i , where $i \in \mathbb{N}$, the AGV's start location s and its goal location g . The AGV is a point-robot, which translates freely at fixed orientation with bounded velocity modulus. The maximum velocity that the robot can reach is denoted by \bar{v}_{\max_R} . Every P_i in W can be either static or linearly moving as well as shrinking or expanding. The velocity of the moving obstacles and the obstacles, which alter their size is less than \bar{v}_{\max_R} . Every moving P_i , before and after its motion has velocity equal to zero. The environment's obstacles are not allowed to come into contact with each other at any time. The problem is to plan a time-minimal semi-free motion for R , from its start point to its goal point, given that the AGV's start and goal points are collision-free at all times and that the descriptions of the obstacles (such as shapes, locations and velocities) are accurately known ahead of planning.

Problem 4 (Obstacles with Piecewise Linear Motion)

Consider the problem of planning the motion of an AGV R in a two-dimensional workspace W populated by simple polygonal obstacles P_i , where $i \in \mathbb{N}$, the AGV's start location s and its goal location g . The AGV is a point-robot, which translates freely at fixed orientation with bounded velocity modulus. The maximum velocity that the robot

can reach is denoted by \bar{v}_{\max_R} . Every P_i in W can be static or moving. The moving obstacles can have either linear motion with constant velocity, which is less than \bar{v}_{\max_R} or piecewise linear motion with constant velocity for each time interval, which is less than \bar{v}_{\max_R} . Note that their velocity in different time intervals does not have to be the same. Every moving P_i , before and after its motion has velocity equal to zero. The environment's obstacles are not allowed to come into contact with each other at any time. The problem is to plan a time-minimal semi-free motion for R , from its start point to its goal point, given that the AGV's start and goal points are collision-free at all times and that the descriptions of the obstacles (such as shapes, locations and velocities) are accurately known ahead of planning.

6.2 Environments with Shrinking and Expanding Obstacles

In this section Problem 3 is studied and the extensibility of the D*MECHA algorithm for solving it is investigated. As was mentioned in section 6.1, there are significant applications, which can be formulated using shrinking and expanding obstacles in their environment. Some, examples of applications whose environments can be described with obstacles, which alter their size over the time, are as follows.

- A real world application which can be modelled as an environment containing shrinking and expanding obstacles is when a robot watercraft is navigating in a sea in which tide gives rise to a difference of the water level on reefs and islands. The immediate affect on the robot's workspace is the shrinkage or expansion of the obstacles. As obstacles are considered the cross sections of the

islands with the plane defined by the sea level. For minimal time vessel routing in time dependent environments and related problems, see (Perakis and Papadakis, 1989) and (Papadakis and Perakis, 1990).

- Another application is when an AGV operates in a manufacturing environment where there are multi-arm stations that operate as part of the manufacturing process. As these stations extend and retract their arms they can be considered as shrinking and expanding obstacles for the AGV, which operates among them as part of the manufacturing process.
- An important application that can be modelled as an environment with shrinking and expanding obstacles is when an AGV is moving in dynamic environments and there are uncertainties on the obstacles' velocities due to control errors if other robots co-operate in the environment and are considered as obstacles for the AGV or due to lack of the knowledge of the environment before planning. The uncertainty of the obstacles' velocity can be incorporated and the obstacles can be modelled as moving shrinking or expanding obstacles. In this way collisions with the environment's obstacles due to their velocity' uncertainties can be avoided. Also the D*MECHA algorithm can be applicable when the information about the problem domain not accurately defined ahead of planning.

As it can be noticed there are important applications whose environments can be described with shrinking and expanding obstacles. Fujimura (1991) considered this

problem and proposed an extension of the accessibility graph algorithm to handle such environments without worsening the algorithm's computational complexity.

6.2.1 Detailed Description of the Shrinking and Expanding Obstacles

As was mentioned in the specification of the problem the obstacles are convex polygons, which can be either static or moving. Note that an obstacle, which does not change its position over the time but it changes its size is also considered as a moving obstacle.

Shrinking and expanding obstacles are defined as two-dimensional simple polygons whose boundary alters over the time. The alteration of the obstacles' size over time does not happen randomly, but in the following manner. Every vertex p_i of the obstacle P is moving with constant velocity along a linear path defined by a fixed point O inside the obstacle and the vertex p_i , as illustrated in Figure 6.1.

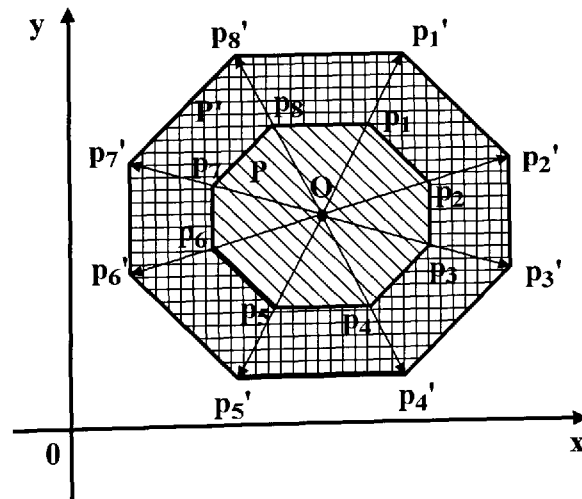


Figure 6.1 Illustration of an obstacle, which expands about its internal point O .

The distance that every vertex p_i of the obstacle covers between two time instances t_α and t_β , with $t_\alpha < t_\beta$, is given by $(c (t_\beta - t_\alpha) \cup p_i)$, where c is a constant which can only take two values, 1 or -1 depending whether the obstacle is shrinking or expanding and \vec{v}_{p_i} is the velocity of vertex p_i . Note that the constant c takes the value -1 , when the obstacle shrinks and the value 1, when the obstacle expands.

Most often the applications require the shrinkage or the expansion of an obstacle to be undertaken uniformly, this means that all vertices p_i have the same velocity and move simultaneously. However notice that the proposed algorithm can handle cases where vertices have different velocities or even move at different times, as long as the obstacle does not deform. That is, if the obstacle is convex it does not become non-convex after the shrinkage or the expansion. Notice at the specification of the problem that the environment's obstacles are not allowed to come in contact with each other at any time therefore the topology of the AGV's free space remains the same at all times.

In this chapter the robot motion planning problem in an environment populated by convex shrinking and expanding obstacles is considered. Intuitively when an obstacle is non-convex the process of shrinkage and expansion is more complicated, simply because one internal point is not always adequate to define the motion of all obstacles' vertices. However notice that the algorithm is still applicable to environments, which contain non-convex obstacles as long as the deformation requirement is preserved.

6.2.2 Construction of the Space-Time Configuration Space

It is known from the specification of the problem that the AGV is a point-robot therefore its configuration space $C = W = \mathbb{R}^2$ and every obstacle's configuration space $CP_i = P_i$. The space-time configuration space $CT = \mathbb{R}^2 \times [0, +\infty)$ and every obstacle's configuration space CP_i maps from C into CT to a prism denoted by CTP_i .

The AGV's space-time configuration space for an environment, which contains shrinking and expanding obstacles, is very similar to the space-time configuration space for an environment, which contains static and linearly moving obstacles. The only difference is that when the environment contains shrinking or expanding obstacles the size of the polygons defined by the cross section of a prism of a shrinking or expanding obstacle and a plane parallel to the x-y plane at two different time instances is not the same. Figures 6.2a and 6.2b illustrate the AGV's configuration space C and its space-time configuration space CT . The AGV's configuration space of Figure 6.1a contains two C-Obstacles. The CP_1 shrinks uniformly over time, with velocity equals to \vec{v}_1 and the CP_2 changes its position over time and it moves in direction d_2 with velocity equal to \vec{v}_2 .

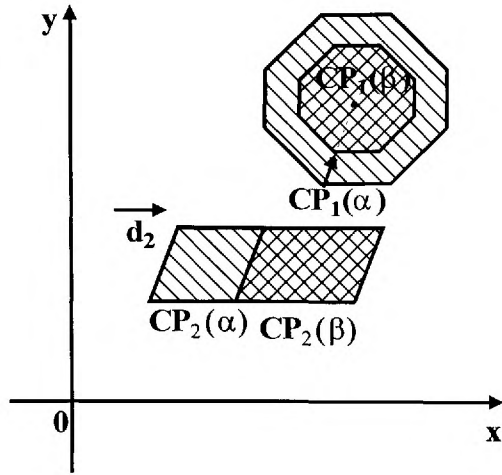


Figure 6.2a The AGV's configuration space containing one linearly moving C-Obstacle and one C-Obstacle which shrinks.

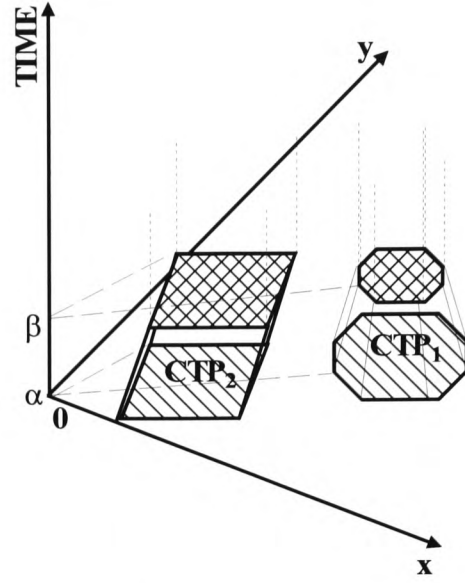


Figure 6.2b The AGV's space-time configuration space for the configuration space of Figure 6.2a.

6.2.3 Applicability of the D*MECHA Algorithm

Recall from the specification of the problem (Problem 3) that the AGV is a point-robot, which is not subject to any kinematic or dynamic constraints. Thus in order to move from its start point to its goal point, attaining a time-minimal motion, it should move constantly with its maximum velocity (\bar{v}_{\max_R}).

In section 5.5 a theorem (Theorem 5.1) was presented, stating that the time-minimal semi-free motion between the AGV's start point and its goal point in an environment populated by stationary and linearly moving obstacles, turns only at obstacles' vertices. Note that this theorem remains valid for environments populated by shrinking and

expanding obstacles and its proof follows immediately once the AGV's space-time configuration space CT is constructed. Therefore the time-minimal motion between two query points in an environment populated by stationary, linearly moving and shrinking and expanding obstacles is contained in the RVG_g in CT.

Note also that Propositions 5.1 and 5.2 presented in section 5.5 that are used for the reduction of the configurations considered for the construction of the RVG_g and the identification of the semi-free motion, are still valid and applicable in environments populated with shrinking and expanding obstacles. Therefore, after the construction of the CT the D*MECHA algorithm can be applied to find the time-minimal, semi-free motion from the AGV's start configuration to its goal configuration. The D*MECHA algorithm was presented in section 5.5 therefore it will not be presented again here.

6.2.4 Sensitivity Analysis of the Admissibility and Optimality Properties of the D*MECHA Algorithm

The computational time and space of the algorithm are not sensitive to the introduction of the new motions (i.e. shrinkage or expansion) of the obstacles. Thus the computational time and space of the algorithm remain $O(n^2 \log n)$ and $O(n^2)$ respectively, where n is the total number of the obstacles' vertices (for an environment with convex obstacles).

Note also that the D*MECHA algorithm still maintains its admissibility and optimality properties. The formal proofs of the algorithm's admissibility and optimality when

there are shrinking and expanding obstacles in the environment remain the same as discussed in section 5.7 and therefore they are omitted from this chapter.

Thus the D*MECHA algorithm is immediately applicable to environments populated by shrinking and expanding obstacles, without any further alterations to the algorithm and most importantly without increase in its computational complexity and loss of its admissibility and optimality properties.

Notice that the Time-Minimal Motion Theorem (theorem 5.2) presented in section 5.8 is still valid in environments, which contain shrinking and expanding obstacles, therefore the motion produced by the D*MECHA algorithm when applied in such environments is time-minimal.

The aforementioned arguments lead to the conclusion that the D*MECHA algorithm is applicable in the extended environment specified in Problem 3 in section 6.1 and establishes the time-minimal semi-free motion for an AGV between its start and goal points. Therefore it can adequately solve Problem 3.

6.3 Environments Containing Piecewise Linearly Moving Obstacles

In this section Problem 4 is studied and the extensibility of the D*MECHA algorithm for solving it is investigated. Fujimura (1991) and Fujimura (1993) considered the same problem and presented an algorithm for solving it, using the concept of the accessibility graph. His algorithm establishes the time-minimal motion in environments populated

by piecewise linearly moving obstacles, in time $O(n^2 \log(mn))$, where n is the total number of the obstacles' vertices and m is the average number of turns made by the obstacles.

In this section an extension of the D*MECHA algorithm will be proposed for solving the encountered problem and a critical evaluation of the approach as well as a comparison with other approaches will be presented.

6.3.1 Space-Time Configuration Space with Piecewise Linearly Moving Obstacles

When the AGV's workspace W is populated by piecewise linearly moving obstacles its corresponding space-time configuration space differs from that which contains linearly moving obstacles. The reason is that an obstacle which has piecewise linear motion, moves for a finite number of time intervals in fixed direction with constant velocity. Therefore its corresponding CT-Obstacle in CT is a sequence of a finite number of prisms each of them bending towards a different direction (depending on the direction of the obstacle's motion at the corresponding time interval), with different slopes to the x-y plane (depending on the obstacle's velocity at the corresponding time interval). Note that after the end of the obstacle's motion the corresponding prism becomes orthogonal to the x-y plane. Figure 6.3a depicts the AGV's configuration space for an environment, which contains one piecewise linearly moving obstacle and Figure 6.3b depicts its corresponding space-time configuration space.

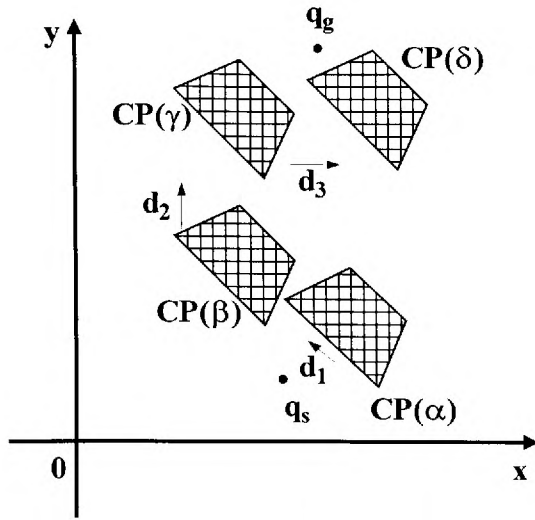


Figure 6.3a The AGV's configuration space C populated by the linearly moving C-Obstacle CP .

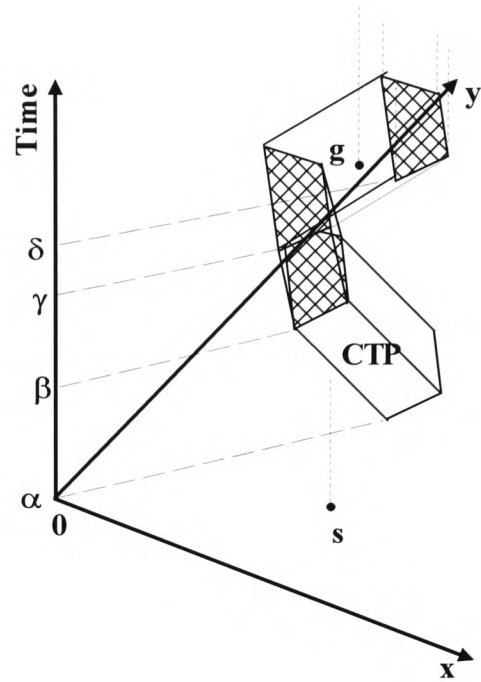


Figure 6.3b The AGV's configuration space-time CT for the C -space of Figure 6.3a.

In the configuration space of Figure 6.3a, the CP is moving within each of the three time intervals (α, β) , (β, γ) and (γ, δ) in directions, d_1 , d_2 and d_3 , with velocities \vec{v}_1 , \vec{v}_2 and \vec{v}_3 respectively.

The CTP in CT consists of a sequence of three prisms each of them bending in the corresponding direction, with the corresponding slope.

It can be seen from Figures 6.3a and 6.3b that all the edges of the CP map into the faces of the prisms (in the sequence) in CT and all the vertices of the CP map into the edges of the prisms, which do not constitute the bases of them, these edges are called *shaft*

edges. The common edges of every pair of adjacent prisms in the sequence are called *common base edges*. The AGV's start and the goal configurations in C correspond to half lines in CT , which emanate from q_s and q_g respectively.

6.3.2 Applicability of the D*MECHA Algorithm

In section 5.5 a theorem was discussed (Theorem 5.1), stating that when the AGV's environment is populated by a set P of linearly moving and stationary simple polygonal obstacles, the time-minimal semi-free motion from the AGV's start point s to its goal point g (providing that one exists), turns only at obstacles' vertices. However as will be suggested by the following example this is not the case when there are piecewise linearly moving obstacles in the environment.

Example

Consider the configuration space of Figure 6.4. The CP is a moving C-Obstacle with piecewise linear motion. Suppose that the CP's initial position is when the C-Obstacle's edge (1, 2) coincides with the line ℓ_1 , as is depicted in Figure 6.4. The AGV's start and goal configurations are q_s and q_g respectively. The CP moves in direction d_1 , with velocity, say 1m/sec. When the edge (1, 2) reaches the line ℓ_2 (this is after 2 seconds) the direction and the velocity of the C-Obstacle changes and it moves in direction d_2 for 10seconds, with velocity, say 0.5m/sec. Suppose that the maximum velocity of the AGV is $\bar{v}_{\max_R} = 1.5\text{m/sec}$. The co-ordinates of the C-Obstacle's vertices when the CP is at its initial position and the AGV's start and goal points are given in Table 6.1.

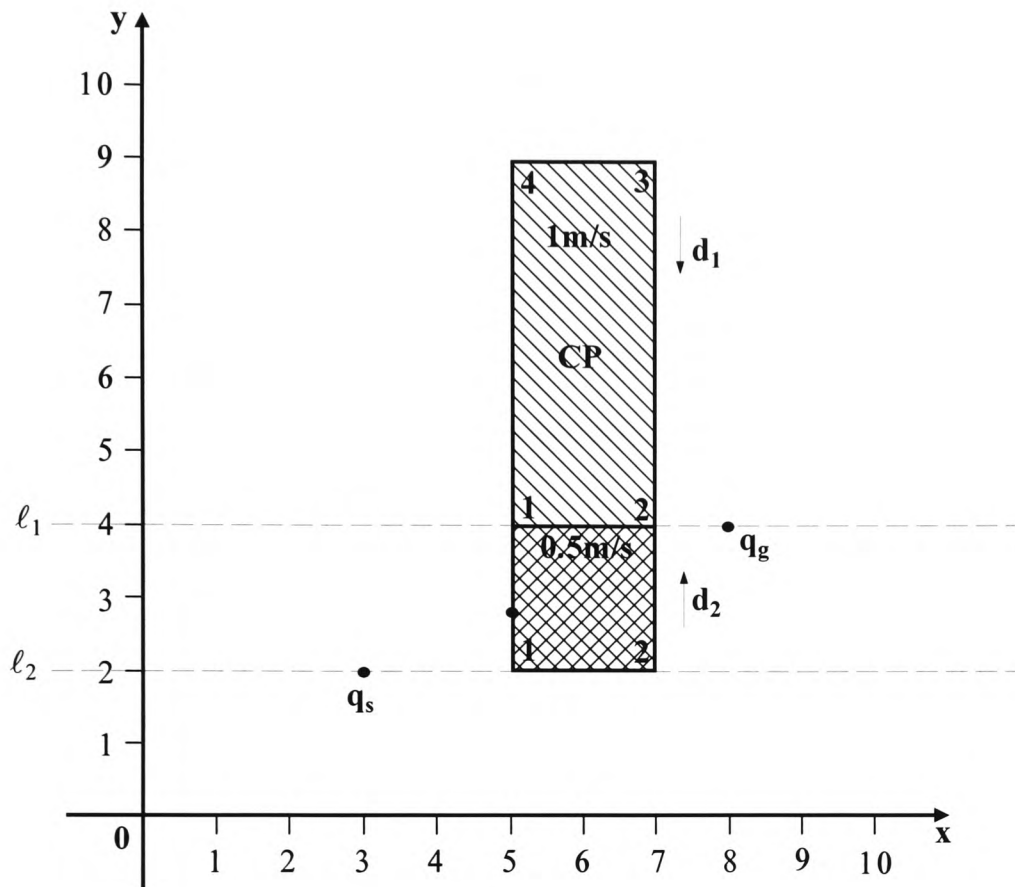


Figure 6.4 An example of a C-space, which contains one piecewise linearly moving C-Obstacle.

Points	Co-ordinates
q_s	(3, 2)
q_g	(8, 4)
1	(5, 4)
2	(7, 4)
3	(5, 9)
4	(7, 9)

Table 6.1 The co-ordinates of the C-Obstacle's vertices (at their initial position) and the AGV's start and goal configurations.

Note that the direction of CP's motion changes from d_1 to d_2 when vertices 1 and 2 have co-ordinates (5, 2) and (7, 2) respectively.

Discussion of the Example

According to the specification of the problem the AGV is a point-robot, which is not subject to any kinematic or dynamic constraints, therefore in order to move from its start point to its goal point attaining a time-minimal semi-free motion it should always move with its maximum velocity, $v_{\max_R} = 1.5 \text{ m/sec}$.

There are two different ways for the AGV to reach q_g from q_s . The first way is to move directly from q_s to q_g , stop and wait at configuration q_α until the CP moves out of its way and then carry on moving towards q_g . This motion is depicted in Figure 6.5. The second way is to move from q_s to q_g by going around the CP, either from below or above. Here it will be shown that neither of these motions is time-minimal and a new way for attaining the time-minimal motion will be presented.

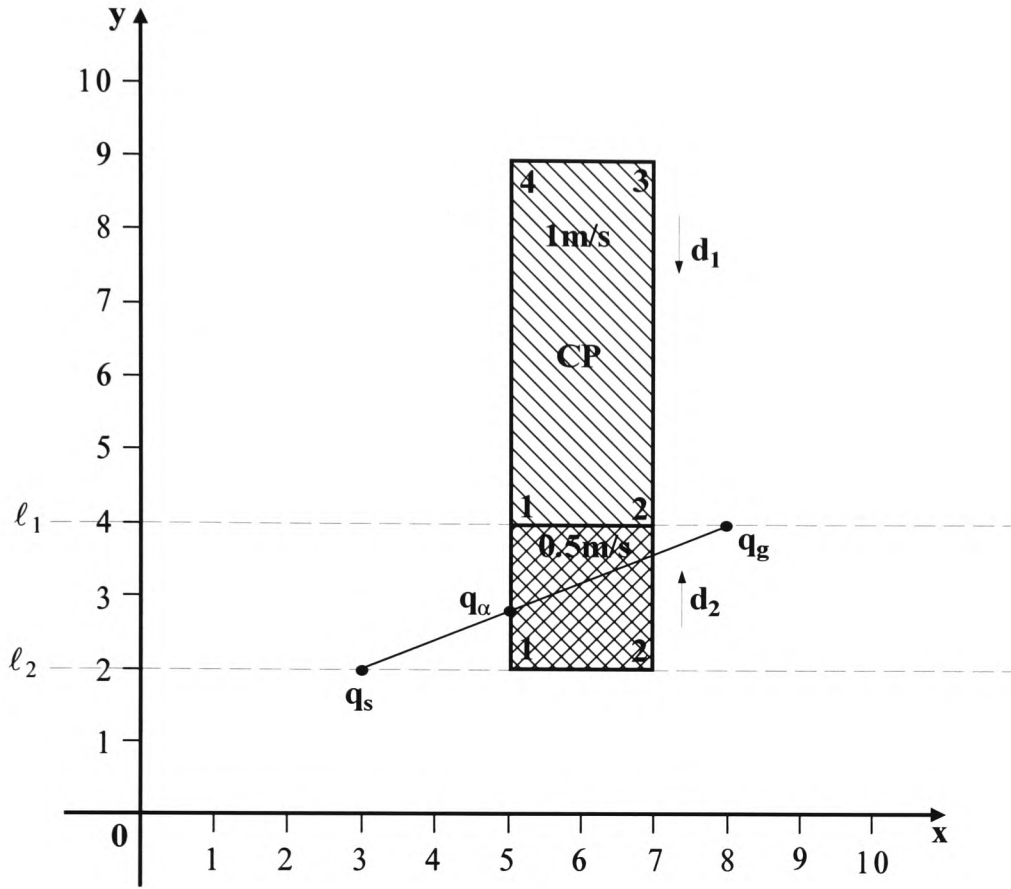


Figure 6.5 Illustration of a stop-motion.

In section 5.8 the *Time-Minimal Motion Theorem* (Theorem 5.2) was presented stating that the time-minimal motion for an AGV R from its start location s to its goal location g in an environment populated by a set P of static and linearly moving obstacles is a non-stop-motion. In section 6.3.5 this theorem will be discussed and it will be shown that it is still valid for environments populated with piecewise linearly moving obstacles. Therefore the former way of reaching the q_g is not time-minimal. The latter way for reaching the q_g is by moving around the CP from either its bottom side or its topside.

If the CP had not changed the direction of its motion after 2 seconds, then if the AGV leaves from the start configuration q_s at time t_0 it could reach the vertex 1 at a configuration q_1 at time t_1 , vertex 2 at configuration q_2 and time t_2 and vertex 4 at configuration q_4 and time t_4 , given that it moves with velocity equals v_{\max_R} Figure 6.6 illustrates these configurations.

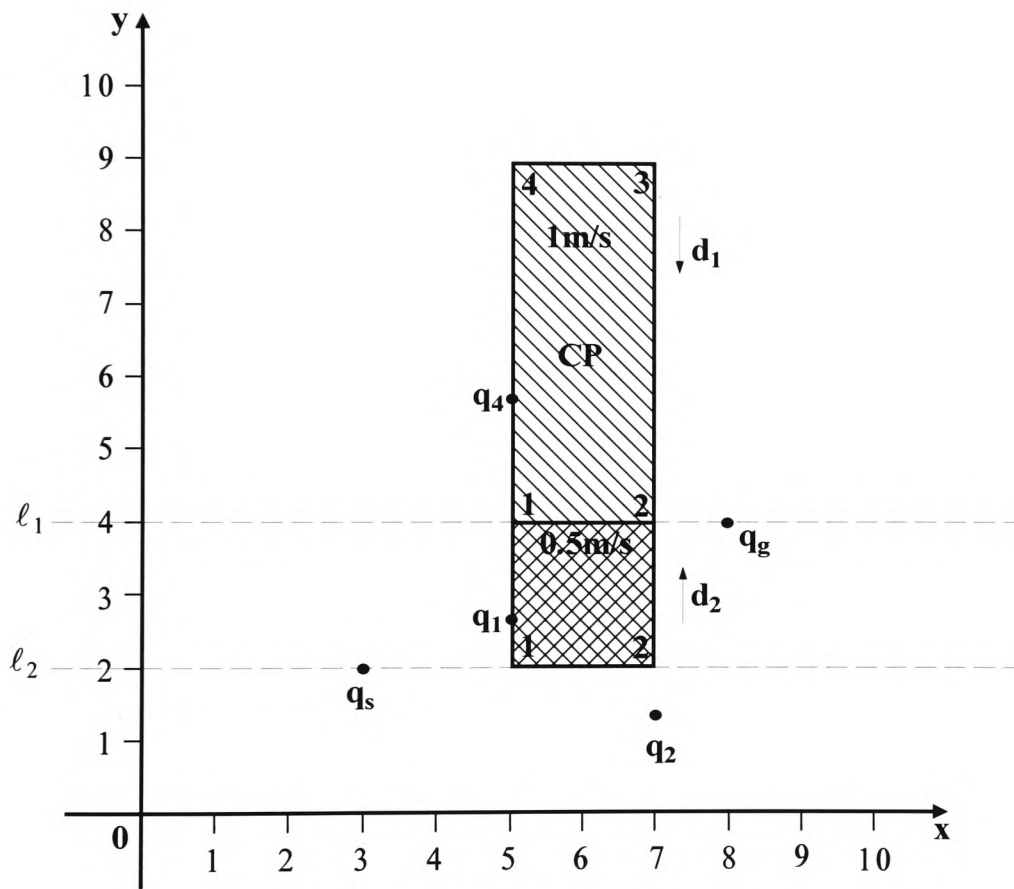


Figure 6.6 Vertices 1, 2, 4 would have been visible and reachable at configurations q_1 , q_2 , q_4 at times t_1 , t_2 and t_4 respectively, if the CP had not changed the direction of its motion when it reached line ℓ_2 .

However the C-Obstacle changes its direction and velocity when its edge (1, 2) reaches the line ℓ_2 . This means that if the AGV starts moving from q_s at time t_0 it will never reach vertex 2 at q_2 , simply because the C-Obstacle will never be at such a configuration, due to the fact that the C-Obstacle changes its direction before vertex 2 is at configuration q_2 . Also note that the AGV will never reach vertex 4 at configuration q_4 , again for the same reason. Notice however in Figure 6.7, that if the AGV departs from q_s at time t_0 , it can reach vertex 1 at configuration q_1 at time t_1 while the CP moves in direction d_1 and vertex 4 at configuration q_4' at time t_4' , after the CP changes its direction and while it is moving in direction d_2 .

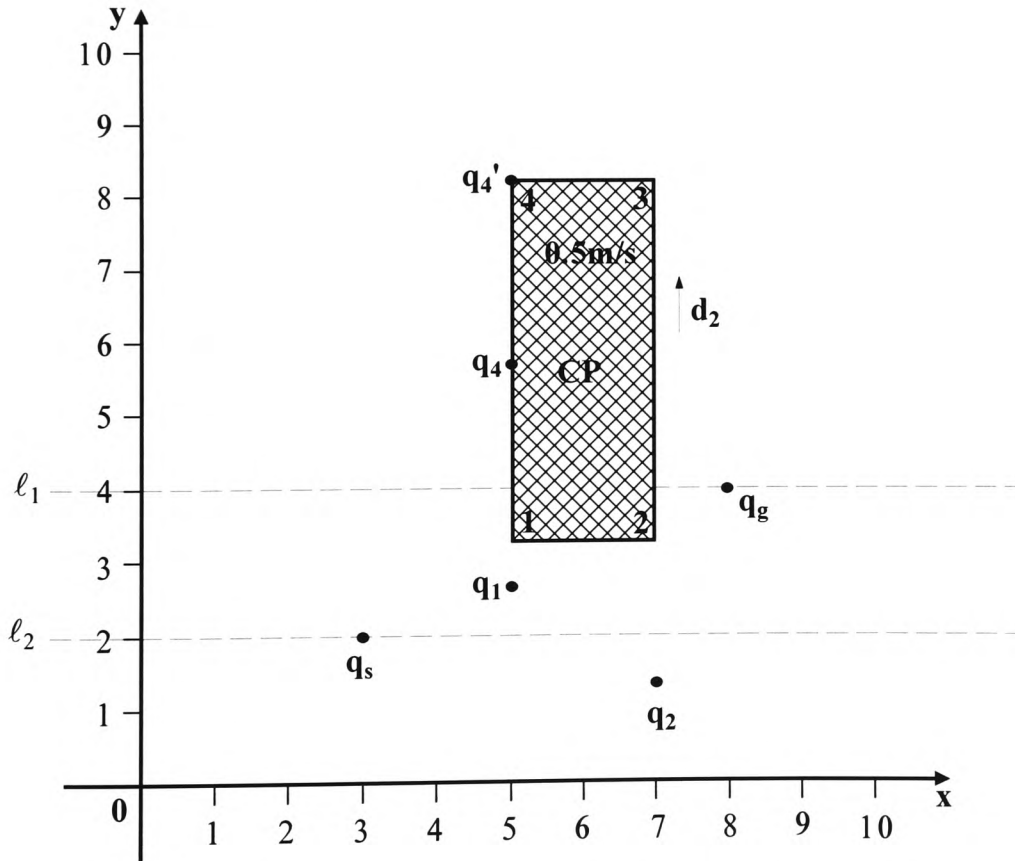


Figure 6.7 Vertices 1 and 4 are visible and reachable from q_s at configurations q_1 and q_4' respectively.

Recall that if the AGV departs from the start configuration q_s at time t_0 it can reach vertex 1 at a configuration q_1 at time t_1 , while the CP is moving in direction d_1 . Thus the AGV could reach vertex 4 at a configuration, say q_4'' at time t_4'' , by going through vertex 1 when is at configuration q_1 at time t_1 , but $t_4' < t_4''$, because the straight line motion is the shortest. Also note that vertex 2 is not visible and reachable from configuration q_1 . Therefore the only way for the AGV to reach configuration q_g from q_s is to go around the topside of the CP, and in particular through configuration q_4' . Figure 6.8 illustrates this motion.

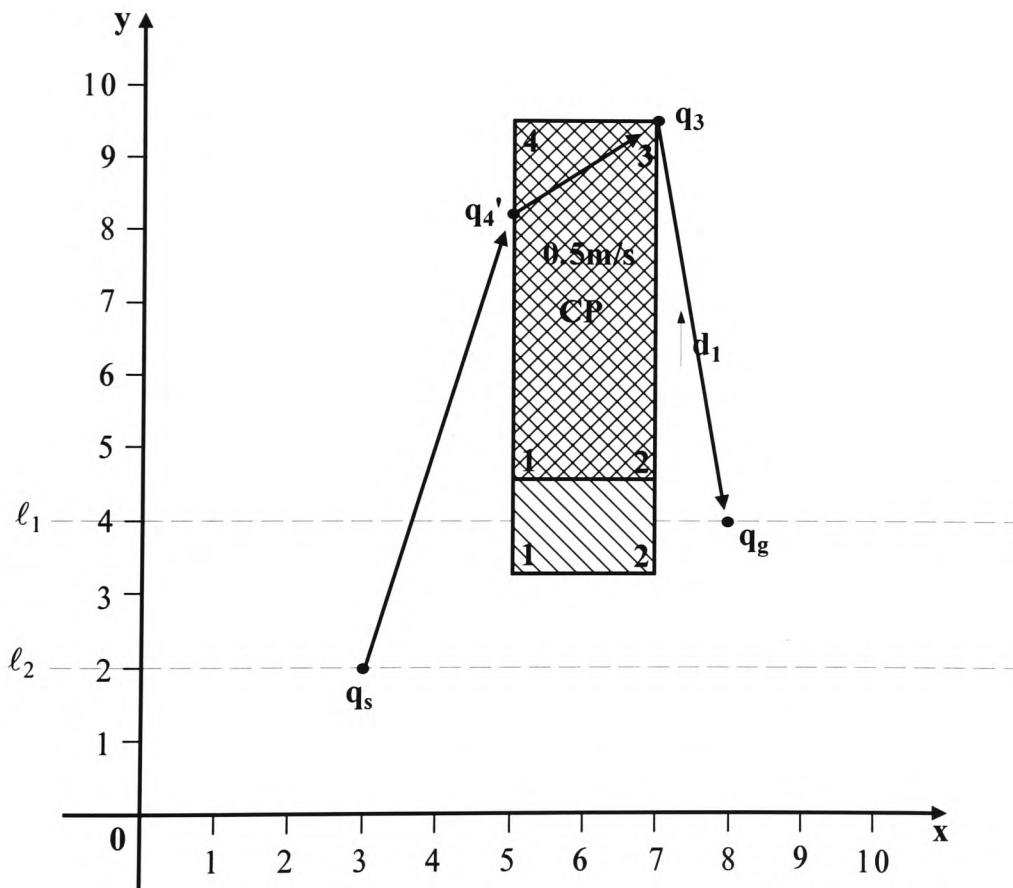


Figure 6.8 The time-minimal motion from q_s to q_g , which goes around the topside of the CP.

The time taken for the AGV to reach configuration q_g from configuration q_s by moving with its maximum velocity along the motion depicted in Figure 6.5 is 9.7seconds.

When the motion of the obstacles is a piecewise linear motion the direction of the obstacles' motion changes over time. Therefore it is possible for the AGV not to reach a vertex of the obstacle due to the fact that the obstacle changed its direction, when it would have reached it if the obstacle had not changed its direction. For instance, in the above example, if the direction of the CP had not changed after 2 seconds, vertex 2 would have been visible and reachable at configuration q_2 .

Vertices 1 and 4 of the CP are visible and reachable from q_s at configuration q_1 at time t_1 and q_4' at time t_4' respectively, however vertex 2 is not. This means that there is always an internal point on the edge (1, 2) of the CP, which lies between vertices 1 and 2 and can serve as a configuration for the AGV to pass through on its way to the goal position. This point is called *assistant configuration*. Note that chronologically, this is the last point of contact between the edge (1, 2) and the AGV. Fujimura (1992) also used the internal assistant point for establishing the time-minimal motion in environments which contain piecewise linearly moving obstacles.

It will be shown here that the motion from q_s to q_g in the above example through such an assistant configuration is less time consuming than the motion, which goes around the topside of the C-Obstacle.

Suppose that an internal point on the edge (1, 2) is visible and reachable from q_s at configuration $q_{ass}^{(1, 2)}$ at time $t_{ass}^{(1, 2)}$. If the AGV departs from q_s at t_0 , it can be at configuration $q_{ass}^{(1, 2)}$ at time $t_{ass}^{(1, 2)}$. If the AGV departs from $q_{ass}^{(1, 2)}$ at time $t_{ass}^{(1, 2)}$ it can meet vertex 2 at configuration q_2'' at time t_2'' , and then it can reach q_g at time t_g . Figure 6.9 depicts this motion.

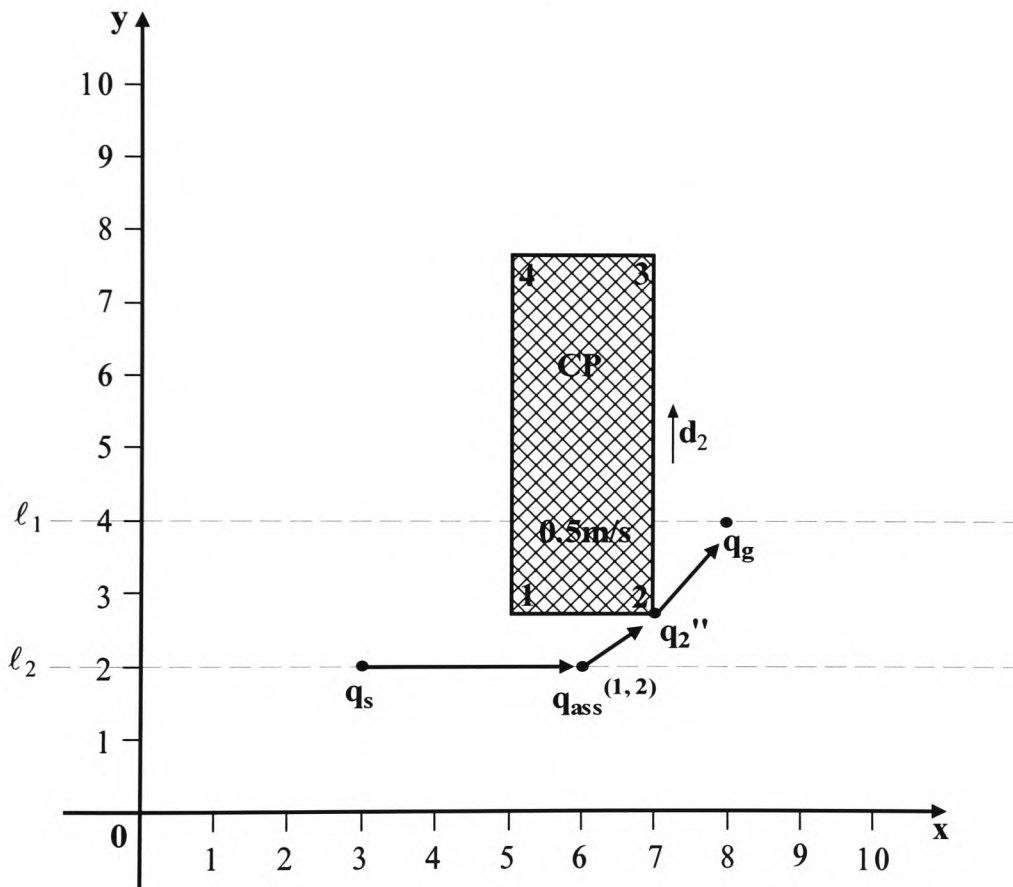


Figure 6.9 A motion from q_s to q_g , through configuration $q_{ass}^{(1, 2)}$, which corresponds to an internal point of the CP's edge (1, 2).

The time taken by the AGV to reach configuration q_g from configuration q_s by moving with its maximum velocity along the motion depicted in Figure 6.9 is 4.05 seconds. It

can be noticed this motion is less time consuming than the motion, which goes around the topside of the CP and its time-cost is 9.7 seconds. Thus the time-minimal motion between q_s and q_g in the given example, is the one, which goes through the assistant configuration $q_{\text{ass}}^{(1,2)}$, which corresponds to an assistant point on the CP's edge (1, 2).

The above example indicates that the time-minimal motion for an AGV (point-robot) between two query points in an environment populated with piecewise linearly moving obstacles can also turn at points other than the obstacles' vertices and in particular at internal edge points of the obstacles. Therefore the Theorem 5.1 discussed in section 5.5 is not valid when the environment contains piecewise linearly moving obstacles.

6.3.3 Extension on the D*MECHA Algorithm

Since the time-minimal motion can turn at internal edge points (assistant configurations) of the piecewise linearly moving obstacles, the process of constructing the RVG_g in the three-dimensional configuration space-time CT, now is somewhat different. The reason for this is because additional configurations other than configurations on the shaft edges of the CT-Obstacles should be considered for the construction of the RVG_g in CT.

The internal obstacles' edges (assistant) points where the time-minimal motion can turn correspond to configurations in CT that are visible and reachable from a configuration, say q , and are on the common base edges between two successive prisms in a sequence of prisms. Suppose that all the assistant configurations need to be identified from a configuration q , in CT. The procedure is similar as the one in section 5.5.2, which identifies visible and reachable configuration in CT, which correspond to C-Obstacles'

vertices in C . Therefore from a configuration q in CT, if a half-line is polar swept about q by keeping a constant angle ϑ (recall from section 5.4 that $\vartheta = \tan^{-1}(\bar{v}_{\max_R}^{-1})$) with the x-y plane, the intersections between the half-line and the common base edges of successive prisms in a sequence, correspond to reachable assistant configurations from q in CT. Once the reachable assistant configurations are identified, then those which satisfy the visibility condition, are retained. Figure 6.10 gives some insight in such configurations.

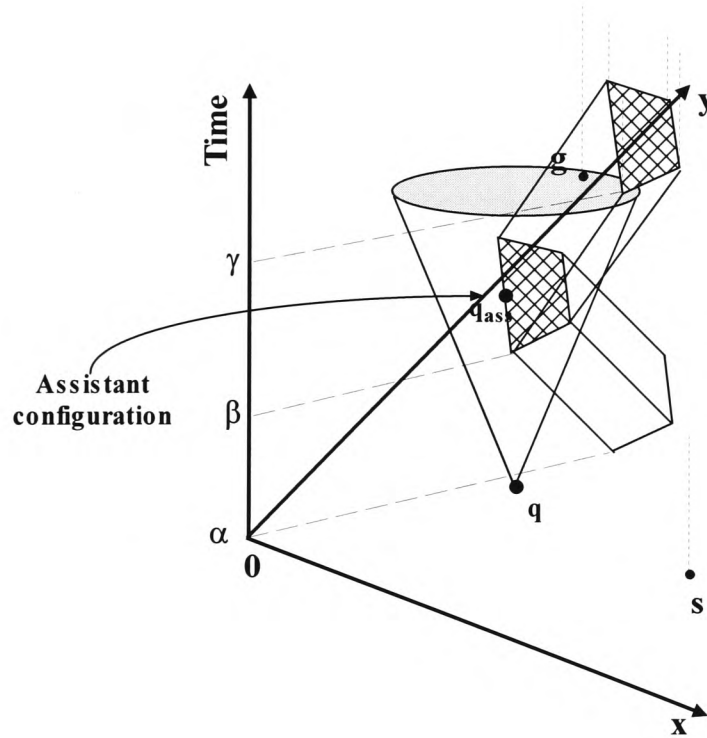


Figure 6.10 Illustration of an assistant configuration (q_{ass}), which is visible and reachable from configuration q in CT.

The algorithm identifies all the visible and reachable q_{ass} configurations from a given configuration q , but for the construction of the RVG_{ϑ} it does not consider those where

both adjacent vertices on the edge are visible and reachable from q . The reason is that if the AGV can reach the two vertices of an edge at some configurations from q , it is less time consuming to move straight to them than passing through an internal assistant configuration. For example consider the configuration space of Figure 6.11.

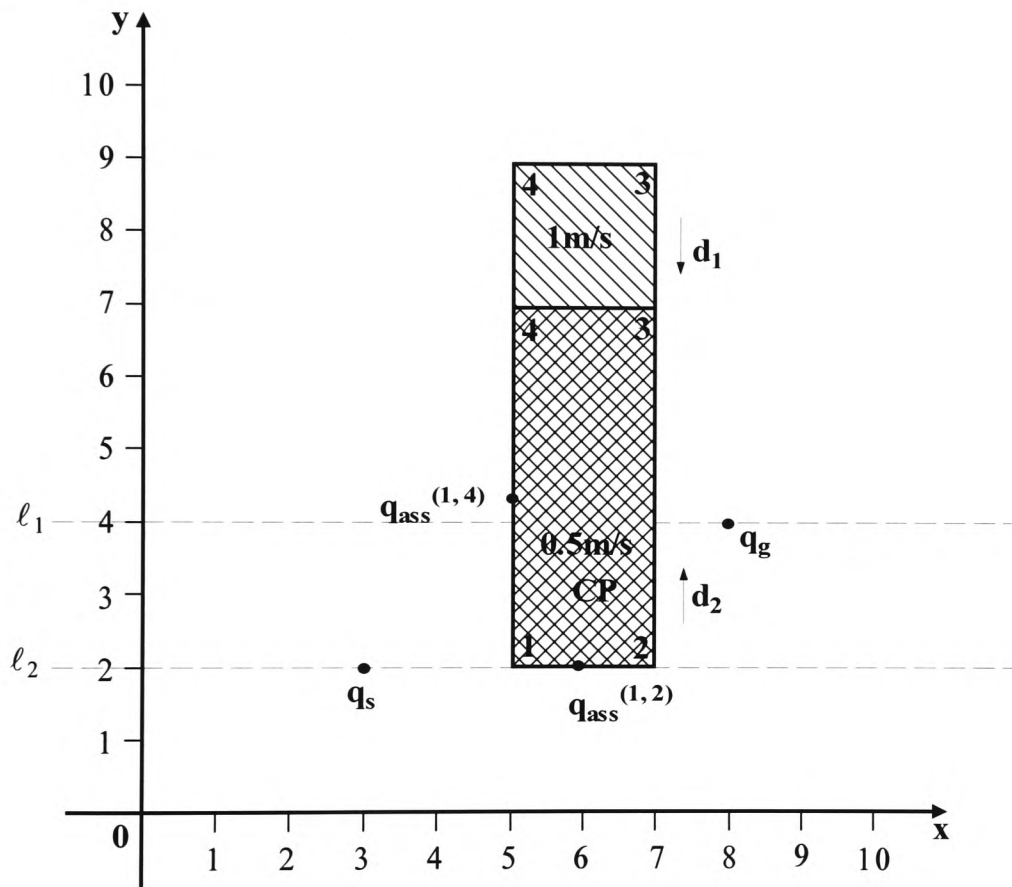


Figure 6.11 Illustration of the internal assistant configurations.

If the AGV departs from q_s at time t_0 it can reach the assistant configurations $q_{\text{ass}}^{(1,4)}$ and $q_{\text{ass}}^{(1,2)}$ at times $t_{\text{ass}}^{(1,4)}$ and $t_{\text{ass}}^{(1,2)}$ respectively, which is equal to 2 seconds. These two configurations are internal edge points of the CP. The assistant configuration $q_{\text{ass}}^{(1,4)}$ is an internal point of the edge (1, 4) of the CP and the assistant configuration $q_{\text{ass}}^{(1,2)}$ is an

internal point of the edge (1, 2). Notice however, that if the AGV departs from q_s at time t_0 it can reach both vertices of the edge (1, 4), vertex 1 at configuration q_1 while the CP moves in direction d_1 and vertex 4 at configuration q_4 '' while the CP moves in direction d_2 . Therefore the assistant configuration $q_{ass}^{(1,4)}$ does not need to be considered for the construction of RVG_g , because both vertices 1 and 4 are visible and reachable at some configurations, and thus the $q_{ass}^{(1,4)}$ is discarded. However, notice that if the AGV departs from q_s at time t_0 it cannot reach both vertices of the edge (1, 2), because vertex 2 is not visible and reachable at any configuration therefore the assistant configuration $q_{ass}^{(1,2)}$ is considered for the construction of the RVG_g . The algorithm proceeds as follows.

When the D*MECHA algorithm identifies all the visible and reachable configurations on the CTP_i 's shaft edges, from a configuration w , it places them on the list VV. Then it places on the list NCV all the configurations which correspond to super-extremes and non-concave vertices from VV. The algorithm then identifies all the assistant configurations from w , but it only places on the list NCV those assistant configurations whose two adjacent vertices are not in VV. The rest of the algorithm remains the same and proceeds as demonstrated in section 5.5.

Note that each vertex of a C-Obstacle is visible and reachable only once in its life-time from a given configuration. Also note that irrespective of how many different motions a C-Obstacle's edge has, it can only have one internal assistant point from a given configuration.

A common base edge in CT can be visible and reachable by many different configurations at different points along its length. Note that all these points have the same reachable time since a common edge is always parallel to the x-y plane in the CT. However, the x-y co-ordinates of the configurations, which correspond to different assistant points on the same common base edge are different. This has as a result, the RVG_g to have arbitrarily many vertices and the search process for a motion to be extremely time consuming. A way to alleviate this problem is to consider only the assistant point, which is closest to the edge's vertex that is not visible and reachable from the current configuration. For clarity consider the configuration space of Figure 6.12.

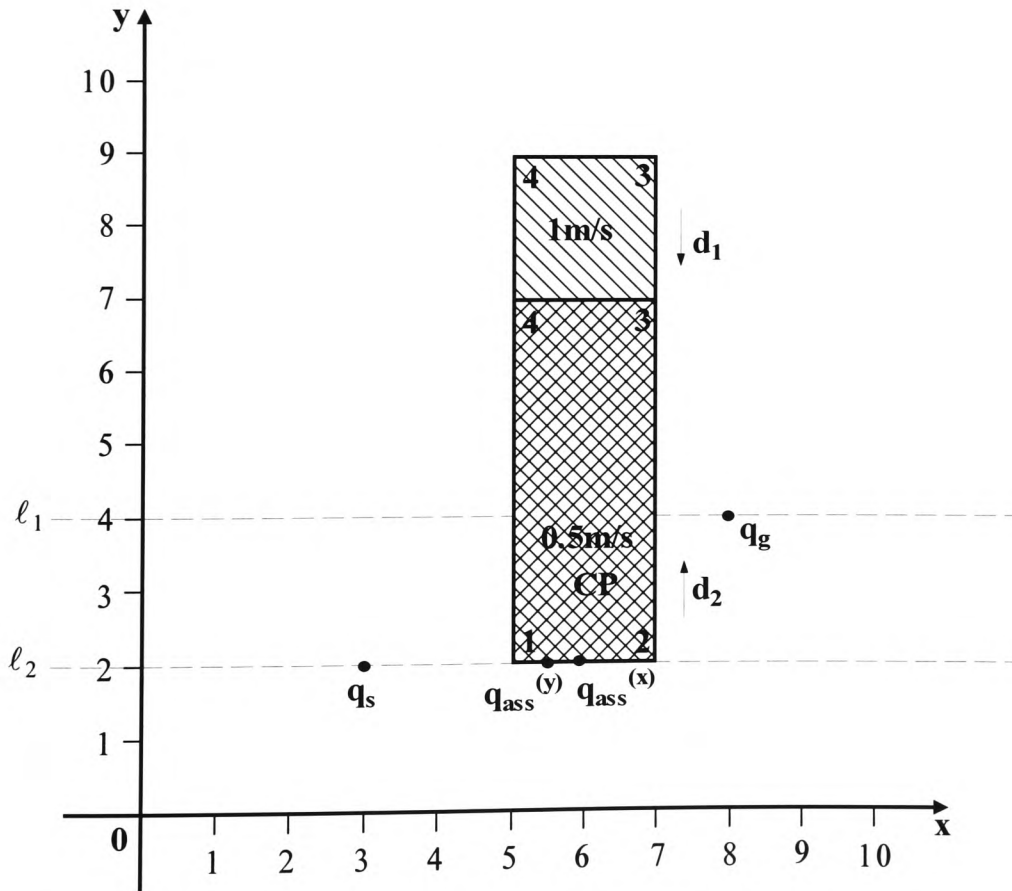


Figure 6.12 Assistant configurations $q_{ass}^{(x)}$ and $q_{ass}^{(y)}$ are on the same edge.

If the AGV departs from q_s at time t_0 it can reach vertex 1 at configuration q_1 and time t_1 but it cannot reach vertex 2 due to the fact that the CP changes its direction before vertex 2 is at such a configuration, which can be reached by the AGV. Therefore an internal assistant configuration on the CP's edge (1, 2) should be considered for the construction of the RVG_g . Note that if the AGV departs from q_s at time t_0 it can reach an internal assistant point on the edge (1, 2) at configuration $q_{ass}^{(x)}$ at time $t_{ass}^{(x)}$ by moving along the straight line motion $\{q_s, q_{ass}^{(x)}\}$. If the AGV departs from q_s at time t_0 it can reach an internal assistant point on the edge (1, 2) at configuration $q_{ass}^{(y)}$ at time $t_{ass}^{(y)}$ (with $t_{ass}^{(y)} = t_{ass}^{(x)}$), by going via configuration q_1 . As can be noticed configurations $q_{ass}^{(x)}$ and $q_{ass}^{(y)}$ are reachable at the same time (this is the time CP changes its direction), but at different locations. Since the assistant configurations are only considered because one of the two vertices of the (1, 2) edge, is not visible and reachable (in this case vertex 2) from q_s , the configuration which is closest to vertex 2 is considered for the construction of the RVG_g . The reason for this is that if the CP had not changed its direction, according to Theorem 5.1 from section 5.5, configuration q_2 (of vertex 2) would have been considered for the identification of the time-minimal motion. However, since this configuration is not visible and reachable from q_s , the configuration on the edge (1, 2), which is closest to vertex 2 should be retained for the construction of the time-minimal motion. Another way to look at it is that since all the assistant configurations on a common base edge have the same reachable time, for the construction of the time-minimal motion only the extreme should be retained.

6.3.4 Analysis of the Time and Space Complexities

As explained in section 6.2.3, making some minor modifications to the D*MECHA algorithm enables it to solve the motion planning problem for an AGV in a two-dimensional environment populated by static, linearly moving and piecewise linearly moving obstacles. Specifically, the algorithm is the same as that presented in section 5.5 but there is one step added. This is the step, which identifies the visible and reachable internal assistant configurations for each currently expanded configuration. In this section an analysis of the modified D*MECHA algorithm is carried out in order to find out whether the modifications influence the computational time and space of the algorithm.

Since the total number of vertices in the Cspace is n the total number of edges in the environment is n as well. Suppose that p is the total number of C-Obstacles in the environment. The average number of edges per C-Obstacle is equal to $e = \frac{n}{p}$, thus

each C-Obstacle has e edges on average. Now suppose m_i is the number of changes in the direction of the i^{th} C-Obstacle. The average number of direction changes for each C-Obstacle is $m = \frac{m_1 + m_2 + \dots + m_p}{p}$ times. Therefore since each C-Obstacle changes

its direction m times on average and it has e edges on average, there will be $(e \cdot m \cdot p)$ common base edges in CT. The visible and reachable internal assistant configuration from each currently expanded configuration w can be identified in $O(m \cdot e \cdot p \cdot \log(m \cdot e \cdot p))$ time, which is equal to $O(nm \log(nm))$, where n is the total number of the C-Obstacles vertices in the Cspace and m is the average number of directions changes for each C-Obstacle. Notice that for every non-concave vertex of a C-Obstacle that is not

reachable and visible from a configuration q , due to the fact that the C-Obstacle changed its direction, an assistant configuration is considered, therefore the number of the executions of the **while** loop of the algorithm remain k , where k is the total number of the C-Obstacles' non-concave vertices. Thus the total time complexity of the algorithm is $O(k(mn \log(mn)))$, where k is the total number of the C-Obstacles' non-concave vertices, n is the total number of vertices in the Cspace and m is the average number of changes in the direction of each C-Obstacle.

The space complexity of the algorithm is $O((k + m p)^2)$, where k is the total number of the C-Obstacles' non-concave vertices, m is the average number of changes in the direction of each C-Obstacle and p is the number of C-Obstacles in the C-space.

6.3.5 The Optimality of the Motion Produced by the D*MECHA Algorithm

Notice that the proof of the modified D*MECHA's admissibility and optimality are the same as in section 5.7. Therefore the modified D*MECHA is admissible, which means that it always produces the time-minimal motion from q_s to q_g within the RVG_g (providing that one exists), otherwise it returns failure. Also the modified D*MECHA algorithm is optimal in the sense that it never expands more configurations than any other algorithm, which is less or equally as informed, for the identification of the time-minimal motion.

It is not hard to extend and prove that the Time-Minimal Motion Theorem (theorem 5.2) presented in section 5.8 is still valid in environments, which contain piecewise linearly moving obstacles. Therefore the time-minimal motion for an AGV between two query

points in an environment, which contains piecewise linearly moving obstacles, is a non-stop motion. In fact, the only difference from the proof presented in section 5.8, is that now the internal assistant points are considered as vertices of the obstacles. Therefore the semi-free motion produced by the modified D*MECHA algorithm when applied in such environments is time-minimal.

6.3.6 Comparison with other Approaches

In this section the D*MECHA algorithm is compared with other popular algorithms for solving the motion planning problem in a two-dimensional environment populated by piecewise linearly moving obstacles. In particular it will be compared with the approaches of Kant and Zucker (1986), Erdmann and Lozano-Pérez (1987) and Fujimura (1991), using the example presented in section 6.3.2.

Recall from section 5.2 that Kant and Zucker (1986), decompose the trajectory planning problem (TPP) into two sub-problems. The path planning problem (PPP) in which a path that avoids collisions with static obstacles is planned and the velocity planning problem (VPP) in which the velocity that avoids collisions with moving obstacles along this path is planned. However, notice that this approach is not very efficient because it will not find a solution when one exists for the case where an obstacle is moving along the path (established in the first stage of the approach) of the AGV, since the AGV cannot alter its path but just its velocity in the second stage of the approach.

In Erdmann and Lozano-Pérez (1987), the configuration space-time was represented as a list of configuration space slices at particular points in time. These times are those at

which a moving object changes its velocity. This approach produces a motion, which consists of a series of straight-line segments between obstacles' nodes in adjacent slices. The robot follows a straight-line motion with constant velocity between the nodes of two slices. Along the path the robot changes its velocity only at nodes of obstacles, when some obstacle's velocity changes. The algorithm runs in time $O(r n^3)$, where n is the total number of the environment's edges and r is the total number of the constructed slices. The algorithm is time resolution-complete between the slices. A way to make the algorithm complete is to introduce slices at the times where the topology of the free space changes.

Fujimura (1991), constructed an accessibility graph in the environment and then searched it for the time-minimal motion between the AGV's start and goal points. Vertices of the accessibility graph are the obstacles vertices at a certain location and time as well as internal points on the obstacles' edges.

Figure 6.13 illustrates the path produced by each of these approaches. The dashed line is the motion produced by Kant and Zucker's approach, the plain line is the motion produced by Erdmann and Lozano-Pérez's approach and the bold line is the motion produced by the D*MECHA algorithm and Fujimura's approach.

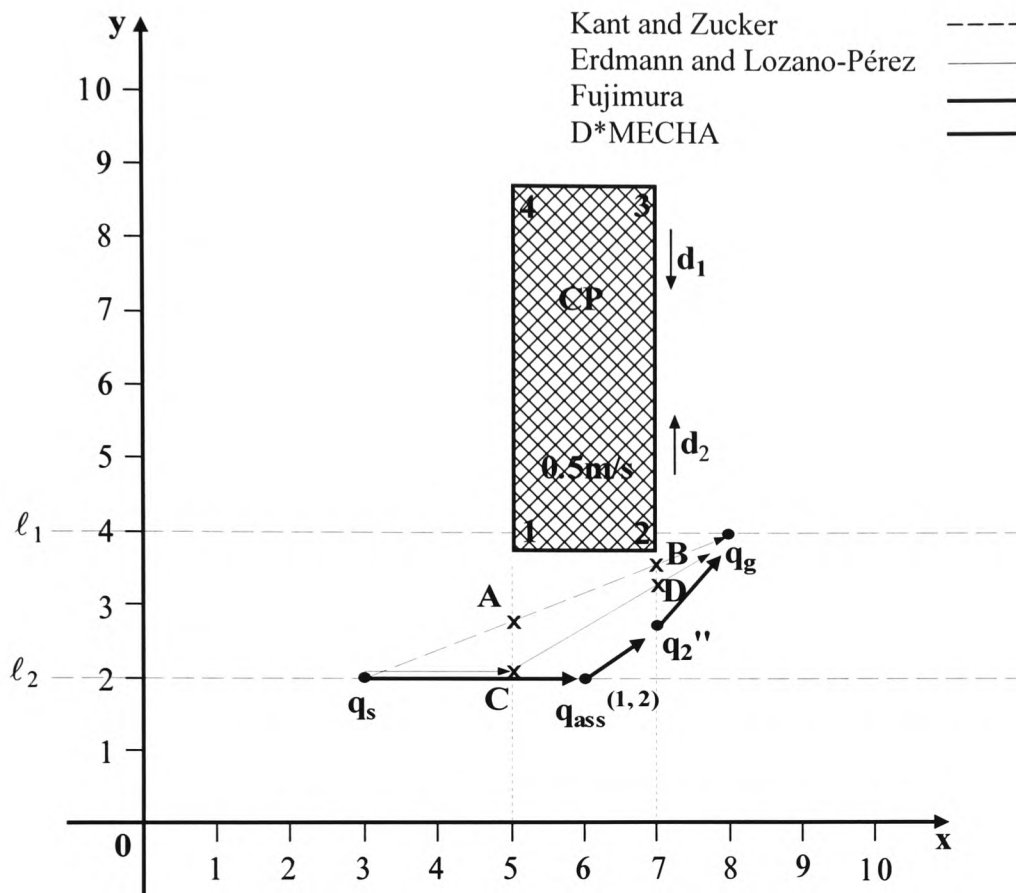


Figure 6.13 Illustration of the four motions.

Note that the motions established by the D*MECHA algorithm and Fujimura's approach are the same, since they are both time-minimal.

Since there is only one CP in the configuration space of Figure 6.13 and this is moving, Kant and Zucker's approach at the first stage, defines as a path from q_s to q_g the straight line-segment, which connects them and then alters the velocity of the AGV along this path in order to avoid collisions with the moving CP. In this approach the AGV starts from configuration q_s at time t_0 and meets with vertex 1 of the CP after 3.66 seconds at the configuration A. From q_s to A, the AGV moves with velocity equal to 0.62m/sec.

The velocity of the AGV then changes to 1.29m/sec until it reaches configuration B (this configuration is where the AGV and vertex 2 of the CP meet). The time-cost of the motion from A to B is 1.66 seconds. Then the velocity of the AGV changes again and the AGV moves from B to q_g at its maximum velocity. The AGV gets from B to q_g within 0.42 seconds. The total time of the motion produced by Kant and Zucker's approach is 5.74 seconds.

In Erdmann and Lozano-Pérez's approach the AGV departs from q_s at time t_0 and moves with velocity equal to 1m/sec to meet vertex 1 of the CP at configuration C. Note that the AGV moves from q_s to C within 2 seconds. The AGV's velocity is then changed to 0.7m/sec until the AGV reaches the q_g configuration. The AGV reaches q_g from C within 4.04 seconds. The total time of the produced motion by Erdmann and Lozano-Pérez's approach is 6.04 seconds.

In Fujimura's approach as well as with the D*MECHA algorithm the AGV is constantly moving at maximum velocity, in this case its maximum velocity is equal to 1.5m/sec. In both approaches the AGV departs from q_s at time t_0 and reaches an internal assistant point of the CP's edge (1, 2), at a configuration $q_{ass}^{(1,2)}$ within 2 seconds. The AGV then moves from $q_{ass}^{(1,2)}$ to configuration q_2'' where it meets vertex 2 of the CP, within 0.66 seconds. The AGV then reaches q_g from q_2'' within 1.39 seconds. The total time of the motion produced by both the D*MECHA algorithm and Fujimura's approach is 4.05 seconds. Table 6.2 summarises the results of the comparison between the four approaches.

Approach	Motion	Velocity m/sec	Time in seconds	Total Time
Kant and Zucker's approach, 1986	$q_s \rightarrow A$	0.62	3.66	5.74
	$A \rightarrow B$	1.29	1.66	
	$B \rightarrow q_g$	1.50	0.42	
Erdmann and Lozano-Pérez, 1987	$q_s \rightarrow C$	1.00	2.00	6.04
	$C \rightarrow q_g$	0.70	4.04	
Fujimura, 1991	$q_s \rightarrow q_{ass}^{(1,2)}$	1.50	2.00	4.05
	$q_{ass}^{(1,2)} \rightarrow q_2''$	1.50	0.66	
	$Q_2'' \rightarrow q_g$	1.50	1.39	
D*MECHA	$q_s \rightarrow q_{ass}^{(1,2)}$	1.50	2.00	4.05
	$q_{ass}^{(1,2)} \rightarrow q_2''$	1.50	0.66	
	$q_2'' \rightarrow q_g$	1.50	1.39	

Table 6.2 Summary of the comparison between the four approaches.

As can be observed in Table 6.2, the motions established by the D*MECHA algorithm and Fujimura's approach for the given example have the same time-cost. This happens because both of the approaches establish the time-minimal motion. The fact that Kant and Zucker's and Erdmann and Lozano-Pérez's approaches produced motions with higher time-cost is an indication that these approaches do not produce time-minimum motions, which is a disadvantage of those approaches over the D*MECHA algorithm and 's approaches.

Figures 6.14a-d illustrate the profiles of the velocity and duration of the motion for each approach.

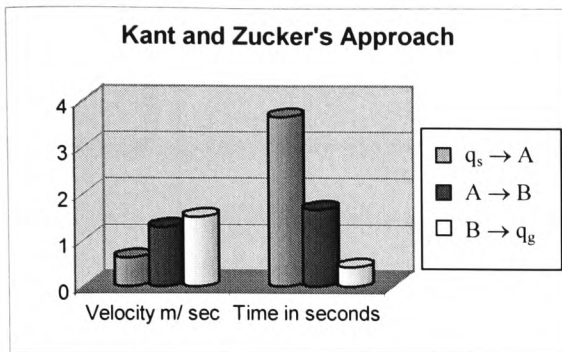


Figure 6.14a Profile of the velocity and the duration of the motion defined by Kant and Zucker's approach.

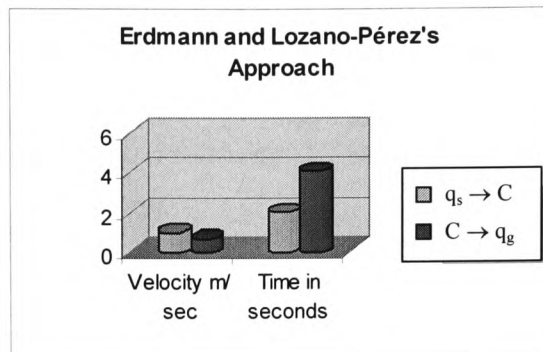


Figure 6.14b Profile of the velocity and the duration of the motion defined by Erdmann and Lozano-Pérez's approach.

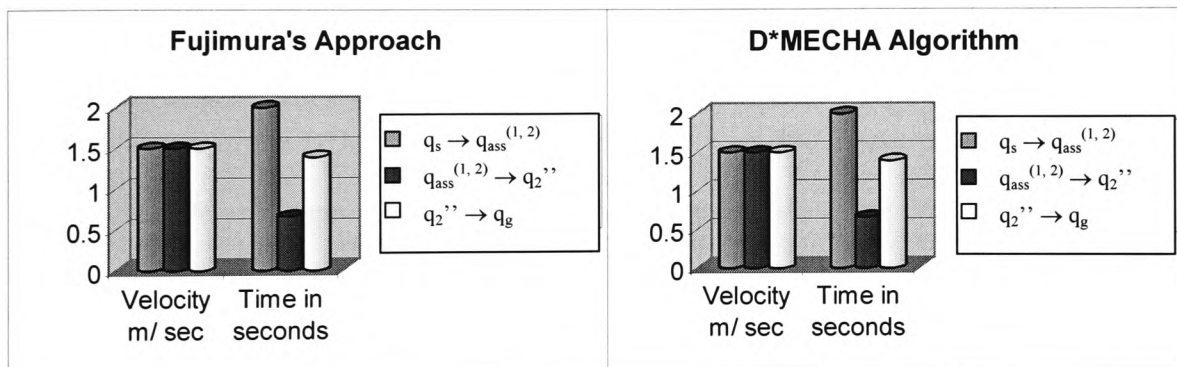


Figure 6.14c Profile of the velocity and the duration of the motion defined by Fujimura's approach.

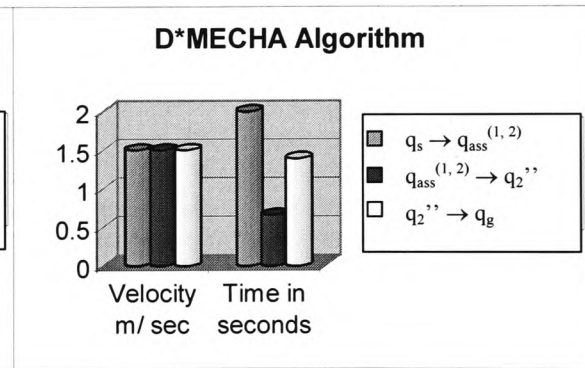


Figure 6.14d Profile of the velocity and the duration of the motion defined by D*MECHA algorithm.

Figure 4.15 depicts the changes in the AGV's velocity while it moves along each of the four motions of Figures 6.13a-d as well as the time consumed by each motion.

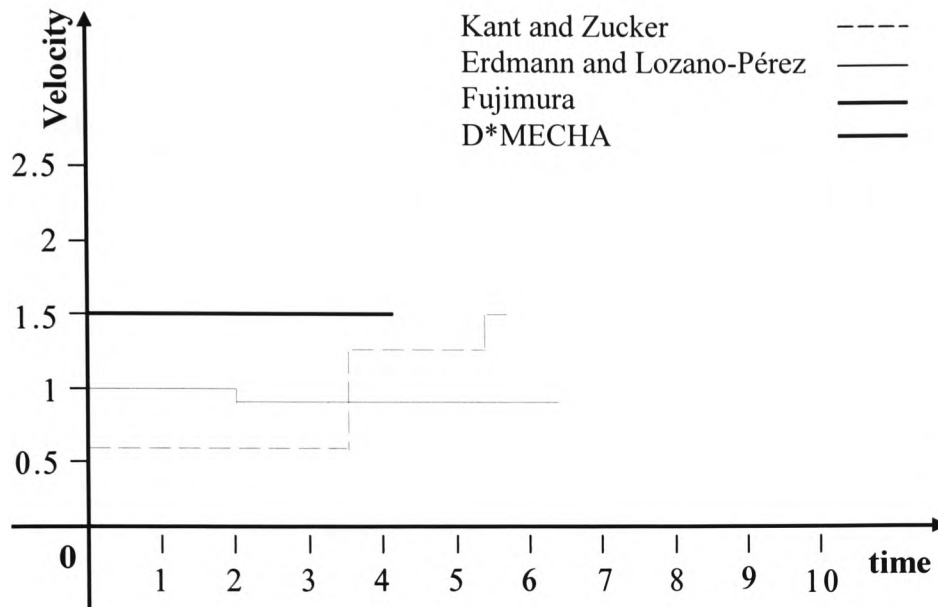


Figure 6.15 Velocity profile of the four motions when used to solve the encountered example.

The advantage of the D*MECHA algorithm over the Kant and Zucker's and Erdmann and Lozano-Pérez's approaches is that in general it establishes time-optimal motions. Also note that Kant and Zucker's approach is not complete while the D*MECHA is. Therefore there could be environments populated by piecewise linearly moving obstacles, where Kant and Zucker's approach fails to find a solution to the motion planning problem but this would not be the case for the D*MECHA algorithm. Note that the D*MECHA algorithm also computationally more efficient than the Erdmann and Lozano-Pérez's approach.

Notice that both the D*MECHA algorithm and Fujimura's approach establish time-minimal motions. Fujimura's approach is computationally more efficient but the author in Fujimura (1991) does not specify whether his approach considers more than one

internal assistant point on a single edge for the construction of the accessibility graph. Thereby this approach can well build an arbitrarily large accessibility graph, which in turn can make the search process extremely time consuming.

6.4 Discussion

In this chapter, two motion planning problems have been considered and the extensibility of the D*MECHA algorithm for solving them was investigated. Specifically the two problems considered were, (i) the motion planning problem for an AGV in a two-dimensional environment populated by shrinking and expanding obstacles and (ii) the motion planning problem for an AGV in a two-dimensional environment populated by static, linearly moving and piecewise linearly moving obstacles.

The importance of both problems was discussed and their applicability was highlighted. The solution to the former problem has great applicability in motion planning for a vessel or other marine vehicles, while it can also be used to formulate the manufacturing environments where articulated arm stations can be modelled as shrinking and expanding obstacles for an AGV, which co-exists and co-operates in the environment, as part of the manufacturing process. With the concept of shrinkage and expansion of the obstacles, problems where there is uncertainty about the obstacles velocity can also be solved. The original obstacles, where uncertainty about their velocity exists can be approximated by shrinking and expanding obstacles, enabling the D*MECHA algorithm to be applicable in somewhat more general environments where there is lack of accurate information about the environment's obstacles ahead of planning.

The applicability of the second problem, which deals with piecewise linearly moving obstacles is more obvious. The reason is that since an AGV operates in a physical environment the formulation of the environment's obstacles as linearly moving objects is not adequate to describe the motion of the obstacles in a general environment. Therefore extending the D*MECHA algorithm to solve the motion planning problem for environments populated by piecewise linearly moving obstacles enabling it to solve more general instances of the classic robot motion planning problem and at the same makes its applicability more realistic.

It was shown that the D*MECHA algorithm can successfully handle environments populated by shrinking and expanding obstacles, thus establishing the time-minimal motion for an AGV, from its start point to its goal point in such an environment, without any extra cost in its computational complexity. It was also shown that the algorithm also maintains its admissibility and optimality properties and in general it establishes optimal motions.

The D*MECHA algorithm with minor changes can also handle environments populated by piecewise linearly moving obstacles. The only modification to the algorithm is that it has to consider internal points on the obstacles' edges as well as their vertices for the construction of the reduced visibility graph. Note that this modification increases the computational time and space of the algorithm to $O(k(mn \log(mn)))$ and $O((k + m)p^2)$ respectively, where k is the total number of the obstacles' non-concave vertices, n is the total number of vertices in the environment and m is the average number of changes in direction of each obstacle.

It was shown that the modified D*MECHA algorithm maintains its admissibility and optimality properties. Thus it always establishes the time-minimal motion for an AGV in a two-dimensional environment populated by static, linearly moving and piecewise linearly moving obstacles.

In conclusion it can be noticed that the flexibility of the D*MECHA algorithm to handle different sorts of environments, with minor modifications, is an indication that the D*MECHA algorithm can solve a number of relevant problems and extensions of the classic robot motion planning problem.

6.5 References

- ERDMANN, M. and LOZANO-PÉREZ, T. 1987. On multiple moving objects. *Algorithmica*, **2** (4), pp. 477-522.
- FUJIMURA, K. 1991. *Motion Planning in Dynamic Environments*. Tokyo: Springer – Verlag.
- FUJIMURA, K. 1993. Motion planning in time-varying domains: the case of cyclic motions. *Advanced Robotics*, **7** (5), pp. 491-505.
- FUJIMURA, K. and SAMET, H. 1993. Planning a Time-Minimal Motion Among Moving Obstacles. *Algorithmica*, **10**, pp. 41-63.

KANT, K. and ZUCKER, S. W. 1986. Toward Efficient Trajectory Planning: The Path-Velocity Decomposition. *The International Journal of Robotics Research*, **5** (3), pp 72-89.

PAPADAKIS, N. A. and PERAKIS, A. N. 1990. Deterministic Minimal Time Vessel Routing, *Operations Research*, **38** (3), pp. 426 – 438.

PERAKIS, A. N. and PAPADAKIS, N. A. 1989. Minimal Time Vessel Routing in a Time-Dependent Environment. *Transportation Science*, **23** (4), pp. 266 – 276.

7

Conclusions and Future Work

Now all has been heard; here is the conclusion of the matter

ECCLESIASTES 12: 3

7.1 Introduction

In this chapter a review of the thesis is presented, some conclusions are drawn, the main contributions of the research are discussed and future work is recommended.

As was mentioned in the introduction of the thesis, the motion planning problem is an everyday task for humans, which they solve relatively easily without any conscious effort. However, the ability that humans possess to plan their own motions or the motions of other objects in a physical environment is very difficult to replicate in a computer program. Motion planning is a very important aspect of robotics and

therefore robust methods and efficient algorithms for solving it are required. The importance of motion planning in robotics arises from the fact that industry requires flexible and autonomous robots, which can plan and execute their own motions and react to the environment. Also the great safety and economic aspects involved in robotic applications contribute towards the need for developing robust and efficient motion planning approaches for robots.

The robot motion planning problem has been shown to be a very challenging problem. The complete approaches currently available for solving it are computationally so expensive that they limit the practicality of the approach and mostly serve as proof of the decidability of the general movers' problem. For this reason much research has concentrated on providing solutions to individual instances of the motion planning problem rather than proposing approaches for solving the general movers' problem.

In this thesis the robot motion planning problem was considered and in particular two instances of it have been studied. The first is the path planning problem for an AGV in two-dimensional static environments. The second is the motion planning for an AGV in two-dimensional dynamic environments. Two approaches and various extensions of them were proposed to provide solutions to these problems.

7.2 Review of the Thesis

In the introductory chapter of the thesis the motivation and the challenges of the robot motion planning were identified and the aim and the objectives of this research were

highlighted. Chapter two is a quick reference of the basic mathematical and algorithmic notions and notations used throughout this thesis.

In chapter three a background literature survey on the robot motion planning problem was conducted and a critical review of the most important approaches for solving it was presented. The advantages and disadvantages of every approach were highlighted and the suitability of each of them in different application domains was discussed. Finally, some key issues about the existing robot motion planning approaches were observed that helped to identify their deficiencies, which in turn later resulted in the proposition of the V*MECHA and D*MECHA algorithms.

Chapter four was concerned with the path planning problem for an autonomously guided vehicle in a static environment populated by polygonal obstacles. Several algorithms based on the visibility graph approach for path planning were briefly discussed and the V*GRAPH (Alexopoulos and Griffin, 1992) was extensively studied. Some methodic and algorithmic deficiencies of the algorithm were identified and recommendations to overcome these deficiencies and complete the algorithm were made. Further recommendations were made to improve the algorithm resulting in the proposition of a new approach (the V*MECHA algorithm) for path planning. The V*MECHA algorithm finds the shortest collision-semi-free path between two query points for an AGV in a two-dimensional environment populated by simple polygonal obstacles. This algorithm is a hybrid approach of the V*GRAPH algorithm and the A* graph searching algorithm. Formal proof of the V*MECHA algorithm's correctness and optimality were described and a critical comparison of it with other similar

approaches was conducted. Finally the advantages and disadvantages of the V*MECHA algorithm over other similar approaches were highlighted and a demonstration of the algorithm through an example was given.

In chapter five the problem of planning a motion for an autonomously guided vehicle in a two-dimensional dynamic environment was considered. Several algorithms, which solve this problem were briefly discussed and the extensibility of the V*MECHA algorithm for solving it was investigated, resulting in the proposition of D*MECHA algorithm. The D*MECHA algorithm is based on the visibility graph approach. It constructs a reduced visibility graph in the AGV's space-time configuration space and then searches it, in order to establish the time-minimal collision-semi-free motion between two query points. The algorithm was demonstrated through an example and formal proofs of its correctness and its optimality were provided. The optimality of the motion produced by the algorithm was also formally proven. Finally, the algorithm was critically compared with other similar approaches and its advantages and disadvantages were reported.

In chapter six several variations of the dynamic robot motion planning problem were considered and the extensibility of the D*MECHA algorithm for solving them was investigated. More specifically, two dynamic robot motion planning problems were studied. The first problem was that of planning a motion for an autonomously guided vehicle in a two-dimensional dynamic environment populated by shrinking and expanding simple polygonal obstacles. The second problem was that of planning a motion for an autonomously guided vehicle in a two-dimensional dynamic environment

populated by piecewise linearly moving simple polygonal obstacles. The motivation for formulating such environments and their applications domain were addressed and extensions to D*MECHA for solving the aforementioned problems were proposed. Proofs of the D*MECHA's correctness and optimality when applied to such environments were discussed. The algorithm was compared with other existing algorithms and its advantages and disadvantages over them were highlighted.

7.3 Discussion and Conclusions of the Research

Motion planning is a fundamental ingredient for the guidance of a robotic system, but as was discussed in section 7.1, complete algorithms for solving it in its full generality are computationally extremely expensive and therefore cannot provide a pragmatic solution in real applications. This justifies the volume of different approaches to the problem to date and the proposition of many different methods for solving it. Due to the fact that the complete methods for solving the generalised movers' problem are computational so expensive, approaches for providing solutions to individual instances of the problem depending on the application domain are proposed. Also heuristic approaches, probabilistic approaches and resolution-complete approaches are employed to make the problem more tractable and its solution more pragmatic and practical but at the expense of the solution's completeness.

The V*MECHA algorithm proposed in this thesis for solving the basic path planning problem is based on the concept of the visibility graph. The reason for this is that the visibility graph approach works quite fast in two-dimensional configuration spaces and

in general establishes optimal paths, which is always desired. Also note that as was demonstrated in chapter six the visibility graph approach can be extended to solve the dynamic motion planning problem without any extra expense in its computational complexity.

The V*MECHA algorithm constructs a reduced visibility graph in the AGV's configuration space and searches it for a path between the AGV's start and goal points. Current visibility graph approaches construct either the entire visibility graph or a reduced visibility graph by only considering the visible common tangents between every pair of obstacles. Rohnert (1986) showed that if there are only convex obstacles in the environment there are at most four such visible common tangents for each pair of obstacles. Liu and Arimoto (1991) and Liu and Arimoto (1992), extended this idea to non-convex obstacles by considering the minimum number of the convex decompositions of the non-convex obstacles for the evaluation of their algorithm's complexity. Their algorithm constructs the T-graph and finds the shortest path within the graph in $O(n(n + t) + m^2 t)$, where n is the total number of the obstacles' convex vertices, m represents the number of the obstacles' convex components and t is the total number of the obstacles' vertices.

The V*MECHA algorithm, using Proposition 4.1 and Proposition 4.2 (see section 4.6) minimises the number of vertices considered for the construction of the visibility graph without sacrificing the optimality of the path. The significant reduction of the size of the visibility graph enables the search process to be carried out more efficiently and effectively, since its efficiency is strongly influenced by the number of edges of the

considered graph. Notice in section 4.12 that the V*MECHA algorithm considers fewer vertices for the construction of the visibility graph than Liu and Arimoto's algorithm resulting in a smaller size visibility graph than the T-graph. Even though Liu and Arimoto's algorithm in general is computational cheaper than the V* MECHA algorithm, as was discussed in section 4.12, there are some occasions where the V*MECHA algorithm attains better computational complexity.

In conclusion notice that although there are algorithms, which construct the entire visibility graph or the T-graph and they are more efficient than the V*MECHA algorithm from a strictly theoretical point of view, in practice most of the time it is advantageous to use the V*MECHA algorithm because the visibility graph it constructs is much smaller and thus the search process for a path is accomplished more quickly.

The D*MECHA algorithm presented in chapter five for solving the dynamic motion planning problem establishes the time-minimal motion. The time-minimality characteristic of the motion produced demonstrates some very interesting properties. These properties enable the algorithm to provide solutions to other related problems. For instance, the algorithm can provide a solution to the decision problem of whether a semi-free motion between the AGV's start point and its goal point that can be accomplished by a given time (deadline) exists. Another problem that can be solved by the D*MECHA algorithm is the one which requires the establishment of the latest departure time for the AGV from its start point in order to arrive at its destination point by a given time. This property is very important when the AGV's motion is part of a manufacturing process in which there are different tasks that have to be co-ordinated.

For the solution of this problem the goal point and the arrival time are set as start point and start time of the AGV respectively and the obstacles have reverse motions. Notice that now the motion finding process is reversed. The time that the AGV arrives at the destination point (the original start point) by following the time-minimal motion is the solution to the problem.

By the time-minimality theorem it was stated that in order for the AGV to attain a time-minimal semi-free motion it should follow a vertex-to-vertex non-stop motion by moving constantly at its maximum velocity. However, this may appear counter-intuitive to a human being. The reason for that is that humans and in general every animal possess the cognitive ability to follow the quick and at the same time safe motion rather than a dangerous fast motion within a physical environment. Another factor is that the AGV is supposed to be a point subject only to velocity bounds while the real physical AGVs are dimensioned objects subject to an acceleration bound and other physical constraints such as friction and inertia. Finally, the assumption that the robot moves with greater velocity than the obstacles is not always realisable in physical environments. Taking into consideration all these factors a different strategy may have to be employed for the solution of the robot motion planning problem.

The D*MECHA algorithm can be extended to solve other motion planning problems. For example, in chapter six some extensions to the D*MECHA algorithm were proposed to solve, (i) the motion planning problem for an AGV in a two-dimensional environment populated by shrinking and expanding obstacles and (ii) the motion planning problem for an AGV in a two-dimensional environment populated by static,

linearly moving and piecewise linearly moving obstacles. The fact that the D*MECHA algorithm is flexible enough to handle different sorts of environments, with only minor modifications, is an indication that it can solve a number of relevant problems and extensions of the classic robot motion planning problem.

7.4 The Main Contributions of the Thesis

In this section the main contributions of the research presented in this thesis are discussed.

1. The V*GRAPH approach proposed by Alexopoulos and Griffin (1992), for solving the basic movers' problem is extensively studied. An algorithmic deficiency other than that presented by Conn *et al* (1997), is identified and reported.

More specifically, in the V*GRAPH algorithm the authors reduced the size of the visibility graph by observing that the shortest semi-free path never visits obstacles' vertices with obtuse interior polygon angle and that it only visits vertices, which are extreme vertices of visible sequences. It was shown by Proposition 4.2 in section 4.6 the shortest path from the AGV's start point to its goal point cannot contain concave vertices (vertices with interior polygon angle greater than π radians), but this does not imply that it cannot contain obtuse non-concave vertices. Thus since the V*GRAPH algorithm rejects obtuse non-concave vertices it might mistakenly miss a path from the AGV's start point to its goal point, which goes through an obtuse non-concave vertex.

Therefore the V*GRAPH algorithm is not complete and thus incorrect. Conn *et al* (1997), proposed a counter-example showing that the V*GRAPH algorithm is incorrect due to the misuse of the A* algorithm. However, they did not provide enough information about the failure of the algorithm and they did not propose any recommendations for the algorithm's completion. In sections 4.4 and 4.5 a full analysis of the algorithm's deficiencies is provided and recommendations to overcome them are proposed.

2. Corrections for the completion of the V*GRAPH algorithm are proposed along with methodic and algorithmic improvements that resulted in the proposition of a new reduced visibility graph approach called the V*MECHA algorithm, for solving the basic movers' problem.

The V*MECHA algorithm corrects the V*GRAPH algorithm by making proper use of the A* algorithm and by identifying (filtering) correctly the vertices to be rejected for reducing the size of the visibility graph. Thus since the V*GRAPH algorithm sometimes fails to solve the path planning problem correctly, while the V*MECHA algorithm always succeeds, it can be concluded that V*GRAPH is a poor method while V*MECHA is a complete algorithm. The V*MECHA algorithm further reduces the number of vertices considered for the construction of the visibility graph by rejecting the non-super-extremes of the extreme vertices of the visible sequences for each obstacle. Therefore the search process for establishing the shortest path between the AGV's start and goal locations is accomplished much more quickly. New lower bounds are also proposed for the V*MECHA algorithm other than that proposed for the

V*GRAPH, based on the observation that the complexity of the algorithm highly depends on the type of the obstacles (convex/non-convex) that exist in the AGV's environment.

3. The proposition of the routine, which identifies the super-extremes of the extreme vertices of the visible sequence for each obstacle.

A novel feature of the V*MECHA algorithm is the rejection of the non-super-extremes of the extreme vertices of the visible sequence for each obstacle. By only considering the super-extremes of the extreme vertices of the visible sequence for each obstacle, the size of the visibility graph constructed by the V*MECHA algorithm is smaller than that constructed by the V*GRAPH. A new routine has been proposed for identifying the super-extremes of the extreme vertices of the visible sequence for each obstacle, which requires dual numbering for the vertices of the obstacles.

4. The proposition of an algorithm called D*MECHA for solving the motion planning problem for an AGV in dynamic environments.

The D*MECHA algorithm is an extension of the V*MECHA. The algorithm considers the space-time configuration space of the AGV and thus reduces the dynamic motion planning problem to that of the static path planning problem. It then constructs a reduced visibility graph within the AGV's space-time configuration space and searches this graph for the shortest path between the AGV's start and goal points. This path corresponds to the time minimal motion for the AGV between its start and goal points.

A theorem was proposed and proved stating that the time-minimal motion for an AGV between two query points in a dynamic polygonal environment is a non-stop vertex-to-vertex motion, providing that the velocity of the AGV is larger than the obstacles' velocity.

5. Investigation and proposition of possible extensions to the D*MECHA algorithm are made, in order for it to be applicable in more complex dynamic environments are made.

Specifically extensions to the D*MECHA algorithm, for solving the motion planning problem for an AGV in dynamic environments populated by obstacles that change their size over time were proposed. Also, extensions to the D*MECHA algorithm, for solving the motion planning problem for an AGV in dynamic environments populated by piecewise linearly moving obstacles were proposed. Note that the time-minimality property of the proposed by the D*MECHA algorithm, motion is preserved in both cases.

6. Finally discussions of formal proofs of the proposed algorithms' correctness and optimality as well as critical comparisons with existing similar algorithms for solving the motion planning problem are conducted.

7.5 Suggestions for Future Work

After the completion of the research the results obtained opened several possible avenues for further work. This work can be summarised in the following three areas.

- Testing of the algorithm's abilities in a real robotic system.
- Identification of possible extensions to the D*MECHA algorithm, to enable it to solve the dynamic motion planning problem in more complex environments.
- Reduction of the algorithm's complexity.

To fully appreciate the proposed algorithms for robot motion planning, it would be valuable to carry out testing on a real robotic system. The use of simulations or by proving the correctness of the algorithms gives a good indication of the abilities of the algorithms but it eliminates the numerous problems, which arise when applied in real world applications.

As shown in chapter six the D*MECHA algorithm can be extended for application in environments populated by obstacles, which change their size over time and in environments populated by obstacles, which have piecewise linear motion. The D*MECHA algorithm can be further extended to handle environments populated by transient obstacles. Transient obstacles are those obstacles, which appear and disappear from the environment over time. There is a large application domain of dynamic environments, which can be modelled using the concept of transient obstacles. An

example of which is when a robot operates in an environment where objects are placed in it or picked up from it by a crane, such as ports, airports and construction sites. Another situation, which could be modelled as a transient obstacle is an area of the robot's environment, which is temporarily hazardous for the robot, such as a slippery area, which becomes clear after some period of time. It is also possible to model a situation where an AGV operates in a manufacturing environment in which articulated robot arms pick and place objects in the environment as part of the manufacturing process and the AGV should complete its tasks avoiding collisions with the environment's transient objects. Note that when transient obstacles populate the AGV's environment, the Time-Minimal Motion Theorem (Theorem 5.2) is no longer valid. The reason for this is that when an obstacle has transient motion it might pay for the AGV to wait at a certain location for the obstacle to disappear and then start moving towards the goal point rather than circumnavigating the obstacle by following a vertex-to-vertex motion.

The D*MECHA algorithm can also be extended to handle environments populated by accelerating and decelerating obstacles. One way to handle such obstacles is to approximate their motion by a sequence of piecewise linear motions. Note that the approximation can be arbitrarily close so the acceleration or the deceleration of the obstacles can be adequately represented by a sequence of piecewise linear motions. This way the D*MECHA algorithm can handle environments, which contain accelerating/decelerating obstacles and most importantly the time-minimality of the produced motion is preserved as long as the AGV moves faster than any obstacle.

So far, only changes on the environment's obstacles motion have been considered, but what about if there are changes and restrictions in the AGV motions? In other words, what if the AGV is not a free-flying object but is a car-like robot subject to kinematic constraints? The proposed algorithm can still be applicable but a way to incorporate the AGV's kinematic constraints should be defined. A possible way to do it is to employ a method similar to that presented in (Jiang *et al*, 1996), (Jiang *et al*, 1999).

Finally, further research can be done to establish lower bounds on the algorithms' time complexity. It was shown in section 4.12 that even though there are algorithms using the concept of visibility graph for solving the basic path planning problem that have lower time complexity bounds than the V*MECHA algorithm sometimes it is beneficial to use the V*MECHA algorithm, because in reality there are cases where it can solve the basic path planning problem more efficiently time-wise. This happens due to the fact that the visibility graph which the V*MECHA algorithm constructs is very small in size. However it is very difficult to accurately evaluate its time complexity due to its heuristic nature. If the time-complexity of the V*MECHA algorithm is reduced it could become a very powerful tool for path planning, combining both low time complexity and its powerful heuristic nature to solve very efficiently the motion planning problem.

7.6 References

ALEXOPOULOS, C. and GRIFFIN, P. 1992. Path Planning for a Mobile Robot. *IEEE Transactions on Systems, Man and Cybernetics*, **22** (2), pp. 318 – 323.

- CONN, R. A., ELENES, J. and KAM, M. 1997. A Counterexample to the Alexopoulos - Griffin Path Planning Algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, Part B, **27** (4), 721 – 723.
- JIANG, K., SENEVIRATNE, L. D. and EARLES, S. W. E. 1996. Motion Planning with Reversal Manoeuvres for a Non-Holonomic Constrained Robot. *Proceedings ImechE, Journal Engineering Manufacture*, **210** (5), pp. 487 - 497.
- JIANG, K., SENEVIRATNE, L. D. and EARLES, S. W. E. 1999. A Shortest Path Based Path Planning Algorithm for Nonholonomic Mobile Robots. *Journal of Intelligent and Robotic Systems*, **24**, pp. 347 - 366.
- LIU, Y. and ARIMOTO, S. 1991. Proposal of Tangent Graph and Extended Tangent Graph for Path Planning of Mobile Robots. *Proceeding of the 1991 IEEE International Conference on Robotics and Automation*. Sacramento - California. pp. 312 - 317.
- LIU, Y. and ARIMOTO, S. 1992. Path Planning Using a Tangent Graph for Mobile Robots Among Polygonal and Curved Obstacles. *The International Journal of Robotics Research*, **11** (14), pp. 376 - 382.
- ROHNERT, H. 1986. Shortest Paths in the Plane with Convex Polygonal Obstacles. *Information Processing Letters*, **23**, pp. 71 - 76.

Appendices

A

**Computation of the
C-Obstacles using
Minkowski Sums**

A.1 Configuration Space of a Robot

In chapters 4 and 5, two algorithms were presented for solving the robot's path planning and robot's motion planning problems respectively. In both chapters the robot (the AGV) was a point-robot. Solving the path (or motion) planning problem for a point robot amongst dimensioned obstacles is easier than solving the same problem for a dimensioned robot amongst dimensioned obstacles but unfortunately it is not realistic, because real robots have dimensions. However, there is a way to reduce the latter problem to the former one using the concept of configuration space.

Let R be a simple polygonal robot operating in a two-dimensional Euclidean environment populated by a set $P = \{P_1, P_2, \dots, P_n\}$ of simple polygonal obstacles, this environment is the robot's workspace W . Suppose that a global co-ordinate frame F_W is embedded in W . Further suppose that an arbitrary point is chosen in the interior of the robot, this point is called the *robot's reference point* r , (note that this point can be exterior to the robot as well) and that a co-ordinate frame F_R is embedded in R with its origin at r . Note that the F_W is a static co-ordinate frame while the F_R is a moving one.

In Figure A.1 the robot R is a triangular robot and in order to make the demonstration of the configuration space more intuitive suppose that the origin of the F_R coincides with the origin of the F_W . Thus the robot's reference point is at $F_R(0, 0)$ and in this case is also at $F_W(0, 0)$. Since R is a rigid body, every point on R has a fixed position with respect F_R but is moving with respect F_W . Suppose that in Figure A.1 the black dot is the reference point r of the robot. A placement of the robot can be specified by stating

the co-ordinates of its reference point. Thus in Figure A.1 robot r is initially at $R(0, 0)$ and then at $R(4, 6)$.

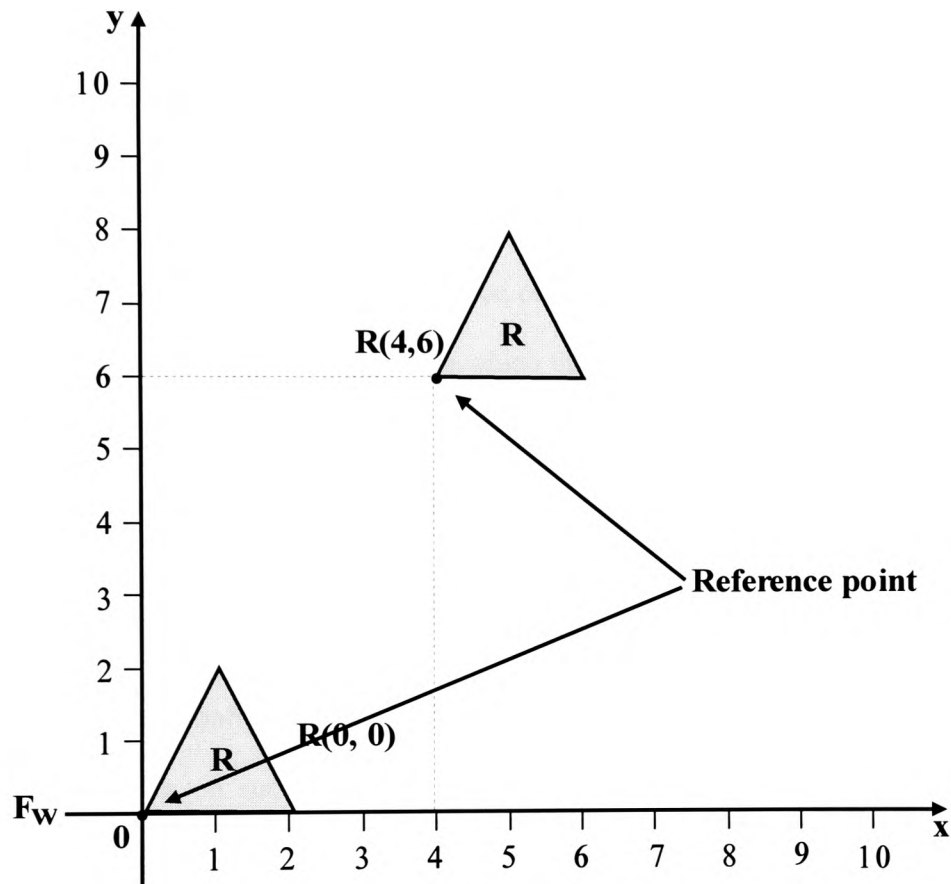


Figure A.1 The placement of a robot can be specified in terms of the co-ordinates of its reference point.

If the robot could rotate as well as translate about its reference point another parameter ϕ is then needed to specify the orientation of the robot. In this case $R(x, y, \phi)$ specifies a placement of the robot R , with its reference point at (x, y) and counter-clockwise rotated about r at angle ϕ with the x -axis of F_w .

A placement of a robot is specified by a number of parameters, which correspond to the number of the robot's *degrees of freedom*. Therefore the number of parameters that are required, to specify a placement of a robot in a three-dimensional space is six, providing that the robot is free to translate and rotate. Three translations and three rotations. The parameter space of R is called *configuration space* or *joint space* and is denoted by *C* or *Cspace*.

It is important to distinguish the workspace of the robot from its configuration space. The workspace is the real world space in which the robot operates. Configuration space is its parameter space. When a robot with two degrees of freedom (two transactional) operates in a two-dimensional environment, its configuration space is identical to its workspace (they are both two-dimensional Euclidean spaces).

Notice that some configurations for the robot in *C* are inaccessible because when its reference point is on such configurations the robot collides with the environment's obstacles. $R(q)$ denotes the subset of *W* occupied by *R* at configuration *q*. Suppose that *W* is populated by a set of obstacles $P = \{P_1, P_2, \dots, P_n\}$. Every P_i maps from *W* into *C* to an area CP_i , which is called *Obstacle's Configuration Space* or *C-Obstacle*. More formally:

$$CP_i = \{q \in C \mid R(q) \cap P_i \neq \emptyset\} \quad (A.1)$$

This area represents the configurations, which are illegal for the robot to attempt, because it will collide with the obstacle P_i . The union of all obstacles' configuration spaces CP_i is called *C-Obstacle region* and is formally defined as:

$$\bigcup_{i=1}^n CP_i \quad (A.2)$$

This region represents all the inaccessible configurations for the robot to attempt, because in these configurations the robot will collide with some obstacles. In section A.2 a method, which uses the Minkowski Sums for computing the C-Obstacle region and thus the collision-free configuration space $C_{\text{free}} = C \setminus \bigcup_{i=1}^n CP_i$ is presented.

A.2 Minkowski Sums

Suppose that R is a two-dimensional, simple, convex polygon, which can only translate in a two-dimensional environment populated by a set P of two-dimensional simple, convex polygonal obstacles. The C-Obstacle, of an obstacle $P_k \in P$ is defined as the set of configurations in C , such that if the robot's reference point is placed at such a configuration, R intersects P_k . Therefore by equation A.1, it is obtained that $CP_k = \{q \in C \mid R(q) \cap P_k \neq \emptyset\}$.

The process of computing the configuration space of the obstacles is called *growing obstacles*. This is because the size of the obstacles is enlarged with respect to the size of the robot and the size of the robot is reduced to a point (point-robot). As long as the point-robot (reference point) is outside of the boundaries of the grown obstacles (C-

Obstacles) the robot lies in a collision-free space. Figure A.2 illustrates the principle for calculating the C-Obstacles when the workspace $W = \mathbb{R}^2$. If P_k is a polygonal obstacle then CP_k is the *grown obstacle* or *Obstacle's Configuration Space* (shaded area) and the robot can now be considered as a point.

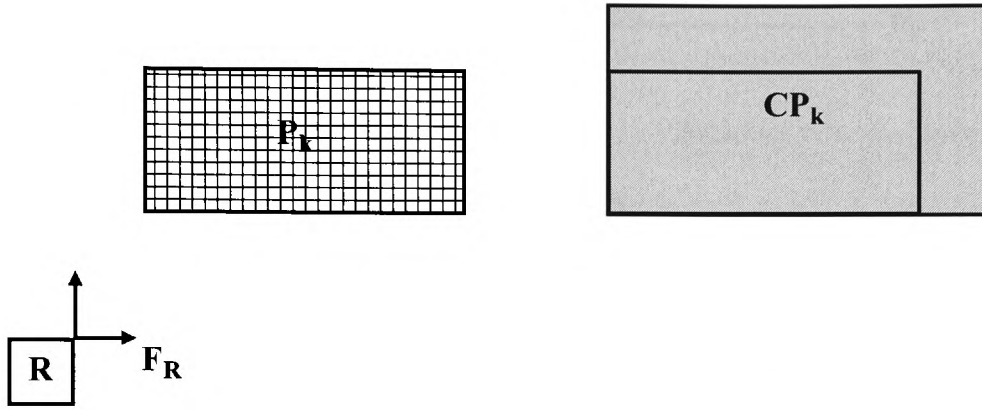


Figure A.2 The shaded areas constitute the grown obstacle or the obstacle's configuration space CP_k .

The above process uses the Minkowski sums to calculate the C-Obstacles. The Minkowski sum of two sets $S_1 \subset \mathbb{R}^2$ and $S_2 \subset \mathbb{R}^2$, denoted by $S_1 \oplus S_2$ and is defined as follows, (Lozano-Peréz and Wesley, 1983),

$$S_1 \oplus S_2 = \{a + b : a \in S_1, b \in S_2\} \quad (A.3)$$

In order to express the C-Obstacles as Minkowski sums one more piece of notation is required. If p is a point $p = (p_x, p_y)$, then $-p$ is defined as $-p = (-p_x, -p_y)$ and for a set S , $-S$ is defined as $-S = \{-q : q \in S\}$. As can be noticed in Figure A3, if S is a set in \mathbb{R}^2 , $-S$, is obtained by reflecting S about its origin.

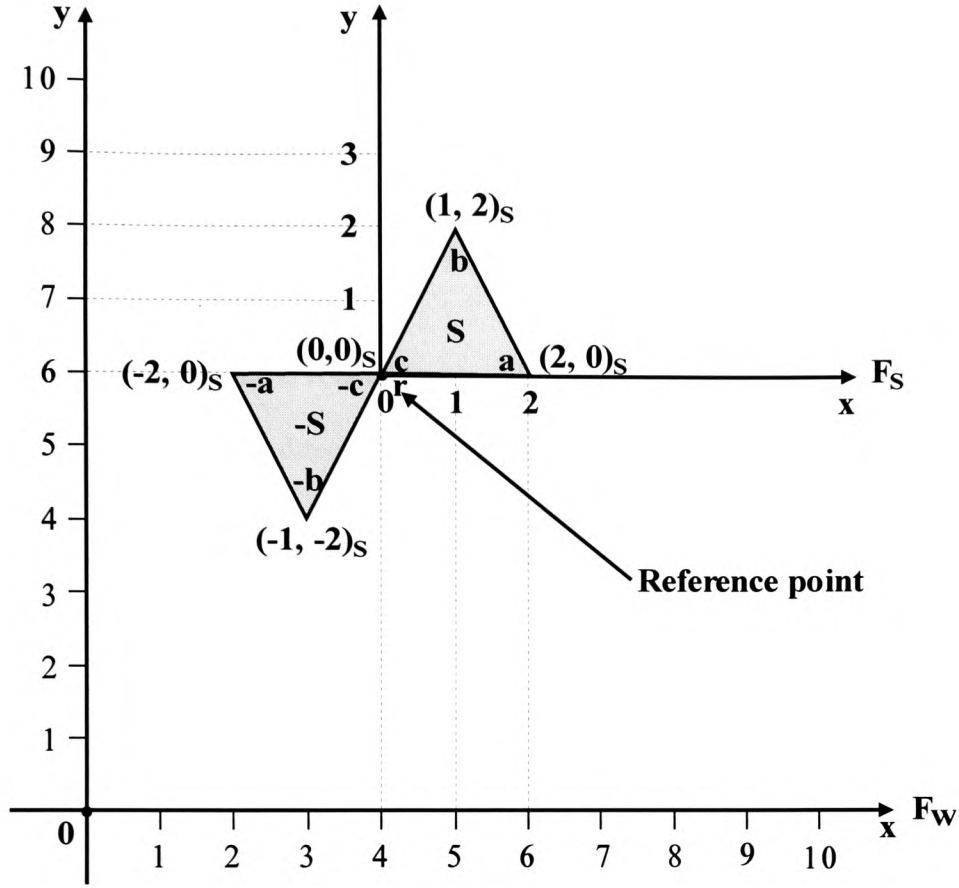


Figure A.3 $-S$ is obtained by reflecting S about its origin.

Theorem A.1

If R is a translating robot in a two-dimensional environment and P_k is an obstacle then the CP_k is $P_k \oplus (-R(0, 0))$.

Proof

For the validity of the theorem what has to be proved is that $R(x, y)$ intersects P_k if and only if $(x, y) \in P_k \oplus (-R(0, 0))$.

Suppose that the robot R intersects P_k when it is at configuration $R(x, y)$ and let q be a configuration in the intersection, with $q = (q_x, q_y)$. Since $q \in R(x, y)$ then $(q_x - x, q_y - y) \in R(0, 0)$ and thus $(-q_x + x, -q_y + y) \in (-R(0, 0))$. However since $q \in P_k$, then $(x, y) \in P_k \oplus (-R(0, 0))$. It is clear that the converse it is also true.

Observe in Figure A.4 that if R and P are two simple convex polygons and $CP = P \oplus R$, an extreme point of CP in direction \vec{u} , is the sum of the extreme points of P and R in direction \vec{u} .

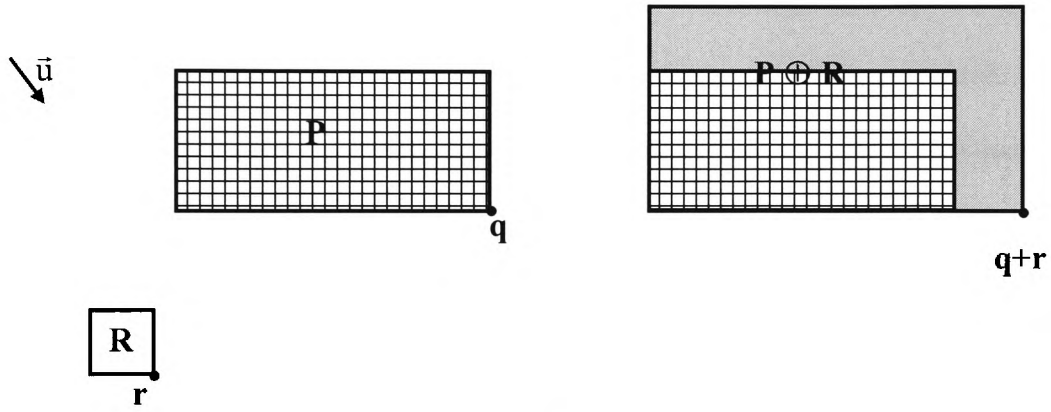


Figure A.4 The extreme point of the $P \oplus R$ in direction \vec{u} is the sum of the extreme point of P and R in direction \vec{u} .

Theorem A.2

If P and R are convex polygons on the plane with n and m edges respectively, then their Minkowski sum $P \oplus R$ is a convex polygon, which has at most $n + m$ edges.

Proof

The convexity of $P \oplus R$ follows immediately from its definition. Observe that in $P \oplus R$ every vertex is obtained when one vertex of each set is combined, and every edge is obtained either when an edge and a vertex are combined or when two edges are combined. The worst case is when P and R do not have any parallel edges because every vertex of one of the sets is combined with every edge of the other set giving rise to maximum $n + m$ edges for $P \oplus R$.

Since the intuitive idea behind the computation of the C-Obstacles using Minkowski sums has been established, an algorithm for the Minkowski sum of two convex polygons can be presented. The ***MINKOWSKI_SUM*** algorithm presented below constructs the $P \oplus R$ by combining vertices that are extremes in the same direction. It is supposed that the vertices are numbered in counter-clockwise order around the obstacles, with r_1 and p_1 the vertices of R and P with the smallest y-value respectively (if there are ties choose the one with the smallest x-value).

MINKOWSKI_SUM

Input : Polygon R with vertices r_1, \dots, r_v and polygon P with vertices p_1, \dots, p_w .

Output : $P \oplus R$

begin

$i := 1;$

$j := 1;$

$r_{n+1} := r_1;$

$r_{m+1} := p_1;$

While ($i \neq n+1$) **and** ($j \neq m+1$) **do**


```

    begin
        Add  $r_i + p_j$  as a vertex in  $P \oplus R$ ;
        If  $\text{angle}(r_i r_{i+1}) < \text{angle}(p_j p_{j+1})$  then
             $i := i+1$ ;
        else If  $\text{angle}(r_i r_{i+1}) > \text{angle}(p_j p_{j+1})$  then
             $m := j+1$ ;
        else begin
             $i := i+1$ ;
             $j := j+1$ ;
        end;
    end;
end.

```

Theorem A.3

If R and P are convex polygons with n and m vertices respectively their Minkowski sum can be computed in $O(n + m)$ time.

Proof

The proof follows immediately by observing that at every traverse of the while loop either i or j or even both are increased by one, until the both reach $n+1$ and $m+1$ respectively.

If R and P are polygons and one of the two is convex and the other is non-convex (say P is non-convex) then P can be triangulated in $m-2$ triangles, where m is the number of its vertices and the union of the Minkowski sums of the convex polygon with every

triangle can be computed in $O(nm)$ time, where n is the number of vertices of the convex Polygon and m is the number of vertices of the non-convex polygon.

In the same manner when both R and P are non-convex polygons their Minkowski sum can be computed in $O(n^2m^2)$ time, where n is the number of vertices of R and m is the number of vertices of P .

B

The A* Algorithm

B.1 Graph Searching

In this appendix the A* search algorithm is discussed and a proof of its admissibility and optimality is provided. Graph searching is in general, a frequent challenge in many computing problems and it is a very important aspect in robots' path and motion planning. There are several techniques for graph searching that have been developed over the years, for example, *breadth-first search* and *depth-first search* (or *backtracking search*). The A* algorithm is a *best-first search* (or *heuristic search*). A* is similar to breadth-first search, with the difference that the search does not proceed uniformly outward from the start vertex, it proceeds biased in favour of some vertices that the heuristic information about the problem domain indicate may lie on the best path to the goal vertex.

B.2 A General Graph Search Approach

The algorithm presented below is a general graph search technique, which starts from the start vertex of the graph and proceeds by applying operators¹ until the goal vertex of the graph is reached.

GeneralGraphSearch

1. **Create** a search graph, G with only the start vertex s. **Put** s on an ordered list OPEN;
2. **Create** an empty list CLOSED;

¹ These operators are functions, which transform one state of the environment into another, which results after an action.

3. **If** OPEN = **nil** **then exit** with failure;
4. **Remove** the first vertex (vertex n) from OPEN and **put** it on CLOSED;
5. **If** n = g (the goal vertex) **exit with success** and **return** the path obtained by tracing the pointers in G from n back to s. (Note that the pointers are created in step 7);
6. **Expand** n, add all its successors in G;
7. **For every** successor j of n **neither** on OPEN **nor** on CLOSED **place** a pointer from j to n and also **put** j on OPEN. **For every** successor j of n **already on either** OPEN **or** CLOSED **if** the newly path to j is less costly than the previously generated **then redirect** its pointer back to n. **If** j is on CLOSED **remove** it and **place** it on OPEN;
8. **Reorder** OPEN according to some criteria;
9. **Goto** 3;

The above process explicitly generates a sub-graph G of an implicitly defined graph. A search tree, ST, subset of G, is defined by the pointers, which are created in step 7. Note that ST is a spanning tree of G. ST is represented by associating to each vertex in G (except s) a pointer to one of its parents. Note that even if G is a general graph step 7 ensures that ST is a tree, since vertex j never records more than one predecessor at the time.

In the algorithm above it is not specified what criteria are used to rearrange OPEN. These criteria can determine in which manner the search proceeds. If the vertices are placed at the end of OPEN in the order in which they generated with no further rearrangement, this leads to a breadth-first search. Note that in this manner the vertices are picked from OPEN in FIFO order. Placing the most recently generated vertex at the

beginning of OPEN with no further rearrangement, leads to a depth-first search. Note that in this manner the vertices are picked from OPEN in LIFO order. These search methods are also called blind-search methods because the choice of which vertex to pick from OPEN to expand next is not influenced by any heuristic factor. In the heuristic search, the list OPEN is rearranged according to the heuristic value of the vertices.

B.3 Heuristic Information

The solution to the graph searching problem offered by the blind-search methods often exceeds the practical limits of computational time and storage. However, it is possible to optimise the computational time and storage factors by taking advantage of available information about the problem graph, to direct the search in the right direction. A way to take advantage of such information is to define an evaluation function, which when applied to every vertex gives an indication of how promising that vertex is, to lead to the goal vertex. This allows the rearrangement of the OPEN in a way such that its elements are ranked in descending order with the most promising vertex to lead to the goal first.

For example, consider the problem of searching a graph, which corresponds to a network of cities, which are connected with intercity roads. The problem requires the establishment of a route between two cities, the start city s and the goal city g . If a blind-search is adopted to solve the problem, the search will proceed from one vertex to another using some predefined rule without taking into consideration any information

about the problem domain, such as how close is a given city to the goal city. However, if some information about the problem domain was available, such as the airline distance between every city and the goal city it could be used to guide the search process, simply by moving to cities closer to the goal city.

Note that the selection of the heuristic information is crucial, it depends on the nature of the problem and differs in different applications. For instance, in the above example of the cities, the distance metric is an adequate heuristic information but in other applications, such as games, puzzles and so on, a different metric may have to be employed so that each configuration can be evaluated based on common features that exhibit with the goal configuration.

B.4 The A* Search Algorithm

The *GeneralGraphSearch* algorithm presented in section B.2 can be a best-first search called A* if an evaluation function \hat{f} is used to minimise the number of vertices visited during the search process ensuring that the best path is obtained. Note that the rearrangement of the vertices on OPEN at step 8 takes place according to increasing values of the function \hat{f} .

Before the evaluation function \hat{f} is defined some additional notation needs to be introduced. Let $g(n)$ be equal to the actual cost of the shortest path between the start vertex s and vertex n , among all possible paths. Further let $h(n)$ be equal to the actual cost of the shortest path between vertex n and the goal vertex, among all possible paths.

Then $f(n) = g(n) + h(n)$ is the actual cost of the shortest path (among all possible paths) from the start vertex s to the goal vertex constrained to pass through vertex n . Notice that $f(s) = h(s)$, since $g(s) = 0$ and is the actual cost of the shortest unconstrained path from the start vertex to the goal vertex.

However, it is not possible to know for every vertex the value of f in advance, therefore an estimation function \hat{f} is used instead. Let \hat{g} be an estimate for g and \hat{h} be an estimate for h , so

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n) \quad (\text{B.1})$$

It is simple to calculate a value for \hat{g} by summing the cost of the arcs along the path already found from s to n . To establish a value for \hat{h} (the heuristic factor) is however more difficult. Therefore for the establishment of its value, heuristic information about the problem domain should be obtained for it to rely upon. In the cities example discussed in section B.3 the airline distance between city n and the goal city can be used as value of \hat{h} . Having defined the evaluation function \hat{f} it is now possible to present the A* algorithm.

The A* Algorithm

1. **Create** a search tree, ST with only the start vertex s . **Calculate** $\hat{f}(s)$ and associated its value with vertex s . **Put** s on an ordered list OPEN;
2. **Create** an empty list CLOSED;
3. **If** OPEN = **nil** **then exit** with failure;

4. **Remove** the first vertex from OPEN and **put** it on CLOSED. **Call** this vertex n ;
5. **If** $n = g$ (the goal vertex) **exit with success** and **return** the path obtained by tracing the pointers in ST from n back to s . (Note that the pointers are created in step 7);
6. **Expand** n , generating a set M of all its successors. **Install** all members of M that are **neither** on OPEN **nor** on CLOSED in the ST as successors of n .
7. **For every** member j of M **do**
 begin
 Calculate $\hat{f}(j)$;
 If j is **neither** on OPEN **nor** on CLOSED **then put** it on OPEN with its \hat{f} value and **place** a pointer from j to n ;
 If j is **already on either** OPEN **or** CLOSED **compare** $\hat{f}(j)$ just calculated with the \hat{f} value previously associated with the vertex and **if** the new value is lower **then do**
 begin
 Substitute it for the old value;
 Redirect its pointer towards n ;
 If j is on CLOSED **then remove** it from CLOSED and **put** it on OPEN;
 end;
 end;
 end;
8. **Reorder** Open in order of increasing \hat{f} values. (Resolve ties always in favour of the goal vertex);
9. **Goto** 3;

B.5 The Admissibility of the A* Algorithm

There are two conditions for graphs and one condition for \hat{h} that guarantee A*'s admissibility, that is when the A* algorithm is applied to such graphs it always finds minimal cost paths. The conditions are:

Condition I: The graph is finite. That is every vertex of the graph has finite number of successors.

Condition II: The graph is a δ graph. That is all arcs in the graph have cost greater than some amount δ .

Condition III: For all vertices in the search graph, $\hat{h}(n) < h(n)$.

In other words it will be shown that if the A* algorithm is applied to a finite δ graph and \hat{h} underestimates h , it always finds minimal cost paths. At the beginning, it will be shown that before the termination of the algorithm, there is always a vertex on OPEN and on the optimal path whose \hat{f} value is less than or equal to the actual cost of the minimal cost path from s to g . Recall that the actual cost of the minimal cost path from s to g is $f(s)$. The aforementioned result is then used to show that the vertex on OPEN with the minimum \hat{f} value cannot be the goal vertex unless its \hat{f} value is equal to $f(s)$ and thus the optimal path has been found.

Lemma B.1

If $\forall n, \hat{h}(n) \leq h(n)$ and the A* algorithm has not terminated, then there is always a vertex n' on OPEN and on the minimal cost path, such that $\hat{f}(n') \leq f(s)$.

Proof

Let the sequence Path = (s, n₁, n₂, ..., g) be the optimal path from vertex s to vertex g. At any time before the algorithm terminates let n'' be the vertex last expanded by the algorithm and n' the vertex with the smallest \hat{f} value on OPEN (the successor of n'' on Path), note that n' could be g. It is known by the definition of \hat{f} that, $\hat{f}(n') = \hat{g}(n') + \hat{h}(n')$, but since all the ancestors of n' have been expanded, the optimal path to n' must have been found, so $\hat{g}(n') = g(n')$ therefore,

$$\hat{f}(n') = g(n') + \hat{h}(n') \quad (\text{B.2})$$

and since it is assumed that, $\hat{h}(n') \leq h(n')$ then,

$$\hat{f}(n') \leq g(n') + h(n') = f(n') \quad (\text{B.3})$$

It is known that the value of the function f for any vertex n on the optimal path is equal to f(s), therefore the inequality B.3 becomes,

$$\hat{f}(n') \leq f(s), \forall n' \in \text{Path} \quad (\text{B.4})$$

and the Lemma is proven.

Theorem 4.2

In a finite δ graph (I, II), if for every vertex $\hat{h}(n) \leq h(n)$ (III), then the A* algorithm is admissible.

Proof

Suppose the contrary, that the algorithm terminates without returning the shortest path between s and g . There are three different cases where this could happen.

1st case: The A* algorithm terminates with a path to a non-goal vertex.

The proof of this contradiction is very trivial and it immediately follows from the termination condition in the algorithm in step 5.

2nd case: The A* algorithm does not terminate at all.

It is not hard to show that the algorithm always terminates. If the A* algorithm does not terminate, it is because it expands vertices on OPEN forever and therefore will expand

vertices in the search tree further than $\frac{f(s)}{\delta}$ steps from s . Since the graph is a finite

graph and is also a δ graph, the \hat{g} values of vertex n on OPEN further than $\frac{f(s)}{\delta}$ steps

from s in the search tree and thus their \hat{f} values will exceed $f(s)$. However, no vertex

further than $\frac{f(s)}{\delta}$ from s is expanded, because it is known by Lemma B.1 that there is

some vertex n' on the optimal path such that $\hat{f}(n') \leq f(s)$, therefore the algorithm will

expand n' rather than n . The only way that the algorithm can now fail to terminate is if

it keeps reopening vertices within $\frac{f(s)}{\delta}$ steps of s . However, each such vertex can be opened only a finite number of times since there are finite numbers of paths for s to n through such vertices within $\frac{f(s)}{\delta}$ steps of s . Therefore the A* algorithm terminates.

3rd case: The A* algorithm terminates with a path to g , however the path is not the shortest.

Suppose that the algorithm terminates at the goal point g but not via the shortest path. From Lemma B.1, it is known that just before the termination of the algorithm a vertex n' on OPEN and on the optimal path existed. Therefore vertex n' would have been chosen for expansion rather than vertex g , which contradicts the assumption that A* terminated.

Having proved all three cases, it is shown that for finite δ graphs, if $\hat{h}(n) \leq h(n)$ then the A* algorithm is admissible.

B.6 The Optimality of the A* Algorithm

As was shown in section B.5, if the three conditions (I, II, III) are satisfied, the A* algorithm is admissible. However, there can be several values of \hat{h} underestimating h . For instance if $\hat{h}(n) = 0$ it is still an underestimate of $h(n)$. Note that if $\hat{h}(n) = 0$ the A* algorithm is a *uniform-cost search* known as the *Dijkstra's shortest path algorithm*.

Consider two different versions of the A* algorithm say A_1^* and A_2^* each of them using a different \hat{h} value, say \hat{h}_1 and \hat{h}_2 respectively. If for all non-goal vertices n , $\hat{h}_1 < \hat{h}_2$, it is then said that A_2^* algorithm is *more informed* than the A_1^* algorithm.

It will be shown here that the A* algorithm is optimal in the sense that it never expands more vertices than any other admissible algorithm, which is less than or equally informed as the A* algorithm, under a restriction on the \hat{h} . The restriction on \hat{h} , is that the difference between the estimated costs from any two neighbouring vertices to the goal vertex g is less than or equal to the cost of the arc connecting the two vertices. More formally for any two neighbouring vertices (when two vertices are neighbouring there is an arc in between them and the cost of the arc is $d_{\text{edge}}(n_i, n_j)$) n_i and n_j ,

$$\hat{h}(n_i) - \hat{h}(n_j) \leq d_{\text{edge}}(n_i, n_j) \quad (\text{B.5})$$

This is called the *consistency assumption* because it must be true when the heuristic information is applied consistently to all vertices.

Lemma B.2

If the consistency assumption is satisfied then for every expanded vertex n by A* $\hat{g}(n) = g(n)$. In other words, this Lemma states that if the consistency assumption is satisfied the A* algorithm has found the optimal path to any vertex n it selects for expansion.

Proof

Suppose that the contrary is true and just before expanding n , that $\hat{g}(n) > g(n)$. Therefore A* has not found the shortest path, but from Lemma B.1 it is known that just before the expansion of n , a vertex n' on OPEN and on the optimal path with $\hat{g}(n') = g(n')$ exists. If $n' = n$ then the Lemma holds, otherwise,

$$g(n) = g(n') + d_{\text{edge}}(n', n) = \hat{g}(n') + d_{\text{edge}}(n', n) \quad (\text{B.6})$$

Under the initial assumption $\hat{g}(n) > g(n)$, so it is obtained that

$$\hat{g}(n) > \hat{g}(n') + d_{\text{edge}}(n', n) \quad (\text{B.7})$$

If the $\hat{h}(n)$ is added to both sides of the inequality (B.7), then

$$\hat{g}(n) + \hat{h}(n) > \hat{g}(n') + d_{\text{edge}}(n', n) + \hat{h}(n) \quad (\text{B.8})$$

By applying the consistency assumption to the right hand side of inequality (B.8) it is obtained,

$$\hat{g}(n) + \hat{h}(n) > \hat{g}(n') + \hat{h}(n') \quad (\text{B.9})$$

Therefore, $\hat{f}(n) > \hat{f}(n')$, which contradicts the fact that the algorithm chose n for expansion while n' was available. Therefore the Lemma is proven.

Lemma B.3

For any vertex n expanded by the A*, if \hat{h} underestimates h then $\hat{f}(n) \leq f(s)$.

Proof

If the vertex n is any expanded by the A* algorithm and n is the goal g then $\hat{f}(n) \leq f(s)$ because the A* algorithm is admissible and the Lemma holds. If n is not the goal then it is known from Lemma B.1 that just before n was expanded there was a vertex n' on OPEN and on the minimal cost path such that $\hat{f}(n') \leq f(s)$. If $n = n'$ then the Lemma holds, otherwise since the algorithm chose n for expansion instead of n' it must have been that,

$$\hat{f}(n) \leq \hat{f}(n') \leq f(s) \quad (\text{B.10})$$

and the Lemma is proven.

Theorem B.2

If the A* algorithm is more informed than another admissible algorithm say, A'*, and the consistency assumption is satisfied, then if a vertex is expanded by A*, it is also expanded by A'*.

Proof

Again suppose the contrary, namely that a vertex n expanded by the A* algorithm but not by A'*. This could happen because some information should be available to A'*

that the cost of a path through n is equally or more expensive than the minimal cost path, so $f(n) \geq f(s)$.

It is known that the actual shortest path from s to g constrained to pass through n is $f(n) = g(n) + h(n)$, rearranging this equation it yields,

$$h(n) = f(n) - g(n) \quad (\text{B.11})$$

However, by A'* it was considered that $f(n) \geq f(s)$, therefore

$$h(n) \geq f(s) - g(n) \quad (\text{B.12})$$

The A'* algorithm could use as a lower heuristic estimate $\hat{h}(n) = f(s) - g(n)$, while the A* algorithm uses as a heuristic estimate $\hat{h}(n) = \hat{f}(n) - \hat{g}(n)$, but from Lemma B.3 it is known that $\hat{f}(n) \leq f(s)$, therefore $\hat{h}(n) \leq \hat{f}(s) - \hat{g}(n)$, meaning that the heuristic factor of A* has satisfied $\hat{h}(n) \leq f(s) - \hat{g}(n)$. However from Lemma B.2, it is known that $\hat{g}(n) = g(n)$, therefore $\hat{h}(n) \leq f(s) - g(n)$. This indicates that the A'* algorithm used information on vertex n , which allowed a lower bound for h , at least as large as the one used by the A* algorithm, contradicting the initial assumption that the A* algorithm is more informed than the A'* algorithm. Therefore the theorem holds.

C

Publications

Appendix C contains all the papers that have been published as a result of the research described in this thesis.

DIAMANTOPOULOS, A., ROBERTS, G. N. and HARWOOD, D. J. 2000. An improved algorithm for finding the time-minimal motion for an AGV in time-dependent environments. *Proceeding of 6th IFAC Symposium on Robot Control, SYROCO 2000*. September 21st – 23rd, 2000. Vienna, Austria. Volume II, pp. 537 – 542.

DIAMANTOPOULOS, A., ROBERTS, G. N. and HARWOOD, D. J. 2000. Extension to the D*MECHA Algorithm. *Proceedings of the 14th International Conference of Systems Engineering*. September 12th – 14th, 2000. Coventry, UK. Volume 1, pp. 119 - 122.

DIAMANTOPOULOS, A., ROBERTS, G. N. and HARWOOD, D. J. 2000. Robot Motion Planning in Environments Populated by Shrinking and Expanding Obstacles. *Proceedings of the 3rd European Advanced Robotics Systems Masterclass and Conference – Robotics 2000 (EUREL)*. April 12th – 14th, 2000. Salford, Manchester, UK. Volume 2.

DIAMANTOPOULOS, A., ROBERTS, G. N. and HARWOOD, D. J. 1999. A heuristic algorithm for motion planning of an AGV in dynamic environments. *Proceedings of the 2nd Workshop on European Scientific and Industrial Collaboration (WESIC 99)*. September 1st – 3rd, 1999. Newport, UK. pp. 207 - 214.

DIAMANTOPOULOS, A., ROBERTS, G. N. and HARWOOD, D. J. 1999. A new hybrid approach for path planning of an AGV. *Proceedings of the 2nd International Symposium Advanced Manufacturing Processes, Systems and Technologies, (AMPST 99)*. March 30th – 31st, 1999. Bradford, UK. pp. 299 – 308.

AN IMPROVED ALGORITHM FOR FINDING THE TIME-MINIMAL MOTION FOR AN AGV IN TIME-DEPENDENT ENVIRONMENTS

A. Diamantopoulos¹, G. N. Roberts¹ and D. J. Harwood²

¹*Mechatronics Research Centre, ²Engineering Department*
University of Wales College, Newport
Allt-yr-yn Campus, PO Box 180, Newport, NP20 5XR, UK
Tel: +44 1633 432487, FAX: +44 1633 432442
e-mail: anastasios.diamantopoulos@newport.ac.uk

Abstract: The use of Autonomously Guided Vehicles (AGVs) or articulated robot arms, is an important consideration for the efficiency of an automated manufacturing process. In the machining industry the use of robotic systems has made the mass production a human unassisted process with very precise results leading to good quality products. In this paper an algorithm for motion planning of an AGV in time-varying environments is presented. More specifically this algorithm finds the time-minimal collision-free motion from an initial point to a goal point for an AGV, in an environment populated by static and time-varying obstacles. The algorithm's computational complexity is $O(n^2 \log n)$, where n is the total number of the obstacles' configuration space vertices. *Copyright © 2000 IFAC*

Keywords: Robotics, AGV, Motion planning, Computational Geometry, Algorithms, Computational complexity.

1. INTRODUCTION

In this paper an algorithm for motion planning of an AGV in time-dependent environments is presented. The algorithm finds the time-minimal collision-free motion from an initial point to a goal point for an AGV in a two-dimensional environment populated by static and moving obstacles. This algorithm is an improvement of the D*MECHA algorithm previously reported by the authors (Diamantopoulos *et al.*, 1999). Some changes have also been made on the D*MECHA algorithm in order to make it more robust. For a survey on motion planning see (Hwang and Ahuja, 1992).

In Diamantopoulos *et al.*, (1999), it was stated that the D*MECHA algorithm establishes a time-minimal motion providing that the AGV is moving in a vertex-to-vertex manner. It was also stated that in the case where the AGV can move in a non-vertex-to vertex manner the motion established by the D*MECHA is greedy in time. This happens because it is possible

for the time-minimal motion, to be defined in such a way that the AGV will have to stop at a specific location to wait for a moving obstacle to move out of its way and then start moving again towards the goal position. However in this paper it is demonstrated that the time-minimal motion between two points, for an AGV in a two-dimensional time-varying environment is a vertex-to-vertex motion, therefore the motion established by the algorithm is time-minimal in general.

2. FORMULATION OF THE PROBLEM

Consider the problem of planning a motion for an AGV A in a two-dimensional workspace W populated by polygonal obstacles P_i , where $i \in \mathbb{N}$, the AGV's start location S and its goal location G. The AGV A is a point-robot, which translates freely at fixed orientation with bounded velocity modulus. The maximum velocity that the robot can reach is denoted by v_{max} . Every P_i in W can be static or moving along

linear paths at fixed orientation, with constant velocity, which is less than v_{\max} . Every moving P_i , before and after its motion has velocity equal to zero. The environment's obstacles are not allowed to come into contact with each other at any time. The problem is to plan a collision-free time-minimal motion for A, from S to G, given that A moves with constant velocity and that the description of the obstacles (such as shapes, locations and velocities) is accurately known ahead of planning.

3. THE CONFIGURATION SPACE-TIME

In time-varying environments the solution of the motion planning problem is not just about spatial reasoning, because the parameter of time should be taken into consideration. Note that in such environments a path could be collision-free in a specific period of time and not free of collisions in a different period of time. Therefore the AGV's configuration space-time CT is defined, by adding the dimension of time in its configuration space C. Its configuration space $C = W = \mathbb{R}^2$ and every obstacle's configuration space $CP_i = P_i$. The AGV's configuration space-time is $CT = C \times [0, +\infty)$ and hence $CT = \mathbb{R}^2 \times [0, +\infty)$. The CP_i map from C into CT as prisms, which are denoted by CTP_i . The static CP_i map into prisms, which are orthogonal to the x-y plane and the moving CP_i map into prisms, which are sloped to the x-y plane in CT. The angle of the slope of any CTP_i is proportional to the corresponding obstacle's constant velocity. Since the boundaries of the CP_i do not come in contact in C at anytime, the boundaries of the CTP_i in CT are not in contact. Figure 1 illustrates the AGV's configuration space C and configuration space-time CT. Note that after the end of each obstacle's motion, the corresponding prism in CT becomes orthogonal to the x-y plane.

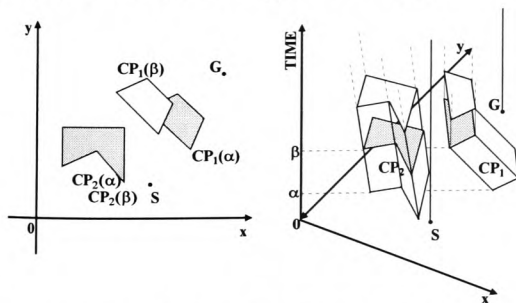


Fig. 1. The AGV's configuration space C and its configuration space-time CT.

From Figure 1, it can be noticed that the edges of the CP_i in C map into the faces of the prisms (CTP_i) in CT. The vertices of the CP_i map into the edges of the prisms, which do not constitute the bases of them, these edges are called shaft edges. The start point S and the goal point G in C correspond to half-lines in CT, which emanate from S and G respectively and are parallel to the time-axis.

With the construction of the AGV's configuration space-time, the problem of planning a motion for an AGV in a time-varying environment has been simplified to that of planning a path for an AGV in a static environment. A path in CT encapsulates both time and location information, therefore since the AGV is moving with constant velocity the Euclidean shortest path from S to G in the three-dimensional CT, corresponds to the time-minimal motion from S to G in C. Once the configuration space-time has been constructed, it can then be searched for a collision-free shortest path from S to G.

4. 'REACHABILITY' AND VISIBILITY

Given that A is moving with constant velocity, the set of all the configurations that can reach between two time instances t_α and t_β from a single configuration p in CT, is defined by the surface of a right circular cone CN, which emanates from p (Figure 2).

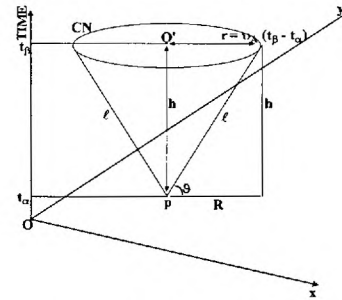


Fig. 2. All the reachable configurations from p are defined by the surface of the cone CN.

The height h of CN is parallel to the time axis and is equal to $t_\beta - t_\alpha$. The radius r of the base of CN is equal to $v_A (t_\beta - t_\alpha)$. The angle θ created by the slant height l of CN and the x-y plane is defined as follows:

$$\tan \theta = \frac{h}{R} \Leftrightarrow \tan \theta = \frac{t_\beta - t_\alpha}{v(t_\beta - t_\alpha)} \Rightarrow \theta = \tan^{-1}(v^{-1}) \quad (1)$$

If CT is polar-swept with a half-line emanating from a point p, at angle θ (with the x-y plane), then all the intersections between the half-line and any of the shaft edges of the prisms correspond to reachable configurations from p. However, in order to say that an AGV is capable of moving from configuration p to configuration q, 'reachability' is not enough. Another condition, which has to be satisfied, is visibility. This means that the two configurations can be connected with an edge and this edge does not overlap the interior of a CTP_i in CT. In summary, if a ray v is swept about p keeping a constant angle $\theta = \tan^{-1}(v^{-1})$ with the x-y plane, all the p-visible configurations at angle θ in the CT can be identified. These p-visible configurations at angle θ in CT correspond to p-visible vertices in C at a specific location at a specific time instance, which can be reached from p by the AGV, given that it is moving with constant velocity.

equal to ψ . In this way a directed graph demonstrating reachability and visibility ('reachability graph' RVG_ψ) at angle ψ can be constructed in CT and then searched for a path. The following propositions are used to minimise the size of RVG_ψ .

Proposition 1

Configurations in CT, which correspond to concave vertices of non-convex obstacles in C, should not be included in the time-minimal motion.

Proof

Consider the environment of Figure 3, the obstacle P is moving in the direction indicated. There are two different ways for the AGV to reach G. The first is to start moving towards G, stop wait at α until the obstacle moves out of its way and then carry on moving towards G, it will be proven in section 6, that this motion is not time minimal. The second one is to go around either side of the obstacle. Assume that the time-minimal motion, from S to G is $\{S, 3, 4, 5, G\}$. This motion goes through the concave vertex 4 of the obstacle P. Since the obstacles are not allowed to come in contact with each other there should be at least a sufficiently small clearance distance σ between them. A point v is chosen between vertex 4 and its successor within distance σ from 4. Note that motion $\{3, v\}$ is collision-free, since the AGV always moves within distance σ from P. It can be shown now, that the motion $\{S, 3, v, 5, G\}$ is less time consuming than the motion $\{S, 3, 4, 5, G\}$, given that the AGV is moving with constant velocity. By using triangle inequalities over the concave vertex, its predecessor and its successor (vertices 4, 3, v respectively), the result obtained is that the motion from 3 to v is less time consuming than the motion from 3 to v through the concave vertex 4. Therefore the motion $\{S, 3, v, 5, G\}$ is less time consuming than the motion $\{S, 3, 4, 5, G\}$, which contradicts the initial assumption that the motion $\{S, 3, 4, 5, G\}$ is time-minimal. This contradiction leads to the conclusion that the time minimal-motion should not contain concave vertices of non-convex obstacles.

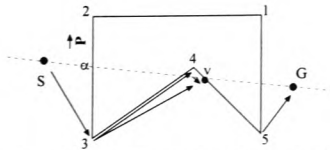


Fig. 3. The time minimal motion does not contain concave vertices.

The algorithm proposed here manipulates concave vertices in the following manner. When an intersection occurs between the sweep line and a shaft edge of a prism, which corresponds to a concave vertex in C, the algorithm rejects it and marks this shaft edge as useless in order that it is not used again. Since this edge is marked as useless intuitively it has to be proven that it should never be used again. If from a configuration w in CT, a configuration, say k_1 is reachable and visible and k_1 corresponds to a

concave vertex k in C, then all the configurations k_m in CT, which correspond to k in C, will never need to be used again. Suppose that a configuration, say k_2 in CT, which corresponds to k in C is reachable and visible from q , then one of k 's predecessors or successors which happen to be a non-concave vertex, or a non-concave vertex of a different obstacle should also be reachable and visible from q . The only case when this is not true is when the environment is bounded by a polygon and there are no obstacles in the environment, but then the time-minimal motion is the straight line connecting S to G. Special attention is needed if the goal point coincides with a concave vertex (of a non-moving obstacle). Using this proposition the problem of taking special care for non-convex obstacles is tackled since the algorithm ignores them.

Proposition 2

A time-minimal motion never visits configurations in CT, which correspond to non-extreme vertices of a visible sequence in C. (Visible sequence is a set, which contains all visible vertices from a given vertex, which are consecutive on a single obstacle's boundary.)

Proof

Consider the environment of Figure 4 the obstacle P is moving in the direction indicated. Assume that the time-minimal motion is the one that goes around the left-hand side of the obstacle, this motion then will definitely pass through vertex 5. Vertices 3, 4, 5, are reachable and visible from S and they are also consecutive on a single obstacle's boundary. The straight-line motion from S to 5 is less time consuming than any other motion, which connects S to 5 and passes through non-extreme vertices of the S-visible sequence (proved by triangle inequalities). Therefore the time-minimal motion should not visit non-extreme S-visible vertices (like vertex 4) of an S-visible sequence. In the same manner the proposition can be proved for the other side of the obstacle, given that the time-minimal motion goes around the right-hand side of the obstacle.

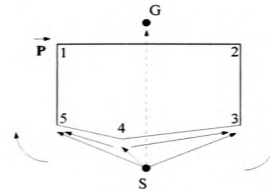


Fig. 4. The time-minimal motion only visits the extreme vertices of any visible sequence.

The algorithm uses propositions 1 and 2 in order to minimize the number of configurations considered for the construction of the RVG_ψ . Propositions 1 and 2 are applied to the two-dimensional projection of the CTP_i in the x-y plane. The use of propositions 1 and 2, in an environment populated by both convex and non-convex moving obstacles, does not reduce the

overall time complexity of the algorithm, but it considerably reduces the size of RVG_g .

5. THE PROPOSED ALGORITHM

The algorithm starts from configuration S and incrementally generates RVG_g , searching it for the shortest Euclidean path between the half lines S and G . This path corresponds to the time-minimal motion from S to G in C . The algorithm is a heuristic search process guided by the evaluation function \hat{t}_f over every configuration. When the algorithm expands a configuration producing more than one candidate configuration for the robot to move toward, the estimation function \hat{t}_f is applied to each of them in order to choose the one with the best assessment. The remaining configurations are stored in a candidate list for later consideration.

The evaluation function \hat{t}_f is defined such that its value $\hat{t}_f(n)$ at any configuration n is an estimation of $t_f(n)$. Where $t_f(n)$ is the time-cost of travelling along the path traced by the actual time-minimal motion from S to G constrained to pass through configuration n . More formally $\hat{t}_f(n) = \hat{t}_g(n) + \hat{t}_h(n)$, where $\hat{t}_g(n)$ estimates the time-cost of travelling from the start location to the current location. The function $\hat{t}_h(n)$ estimates the time-cost of travelling from the current location to the goal location. An obvious choice for $\hat{t}_g(n)$ is the time-cost of the path from S to n the algorithm found so far. A choice for $\hat{t}_h(n)$ is not so obvious, therefore information about the problem domain should be taken into consideration. The airline distance between the current configuration n and the goal half-line can be chosen as a heuristic estimate. Note that the distance to the goal half-line should be measured from the current configuration n to a configuration o on the goal half-line in such a way that the line-segment \overline{no} creates an angle θ with the x - y plane. The search in CT can be carried out either based on information about the Euclidean distance or on information about the travel time (distance over speed). In this approach information about the travel time, has been chosen for the search of RVG_g . The travel time along the airline distance from n to the goal G at angle θ is the fastest possible travel time, so $\hat{t}_h(n)$ underestimates $t_h(n)$. If for all n , $\hat{t}_h(n) \leq t_h(n)$ the algorithm is admissible, (Hart *et al*, 1968). The proposed algorithm finds the time-minimal motion and is stated as follows.

```

begin
   $\theta := \arctan(v_{\max}^{-1})$ 
  put  $S$  in Motion;
  put  $S$  in Open;
  mark  $S$  visited;
   $\hat{t}_g(S) := 0$ ;

```

```

while (Open  $\neq$  nil) do
  begin
     $w := \{i \in \text{Open} : \hat{t}_f(i) < \hat{t}_f(j) \mid \forall j \in \text{Open},$ 
      resolve ties arbitrarily but always favour  $G\}$ ;
    remove  $w$  from Open;
    if  $w = G$  then exit while loop;
     $VV := \{w\text{-visible points at angle } \theta\}$ ;
     $EV := \{\text{extreme vertices of the } w\text{-visible}$ 
      sequences in the projections of  $VV$  into  $Q\}$ ;
    mark all the non-extremes vertices useless
     $AV := EV - \{\text{obtuse vertices in } EV\}$ ;
    for each vertex  $i \in AV$  do
      if  $i$  is not marked useless then
        if  $i$  is not marked visited then
          begin
             $\hat{t}_h(i) := t(d_{\text{air}}(i, G))_\theta$  (the time airline
              distance from  $i$  to  $G$ );
             $\hat{t}_g(i) := \hat{t}_g(w) + t(d(w, i))_\theta$ ;
             $\hat{t}_f(i) := \hat{t}_g(i) + \hat{t}_h(i)$ ;
            put  $i$  in Motion with pointer toward  $w$ ;
            put  $i$  in Open;
            mark  $i$  visited;
          end;
        else if  $\hat{t}_f(i) > \hat{t}_g(w) + t(d(w, i))_\theta +$ 
           $t(d_{\text{air}}(i, G))_\theta$  then
          begin
            redirect pointer of  $i$  toward  $w$  in
              Motion;
            if  $i \in \text{Open}$  then remove  $i$  from Open
             $\hat{t}_g(i) := \hat{t}_g(w) + t(d(w, i))_\theta$ ;
             $\hat{t}_f(i) := \hat{t}_g(i) + \hat{t}_h(i)$ ;
            put  $i$  in Open;
          end;
        end;
      if (Open  $\neq$  nil) then return Motion by tracing all
        the pointers backward from  $G$  to  $S$ 
      else return failure;
    end.

```

In the above algorithm, Motion is a spanning tree, which represents at any instant the best motion obtained so far. For each visited configuration n (except S) a pointer to its parent is held. The function $\hat{t}_f(n)$ is associated with each configuration n in the current Motion. The $t(d(w, n))_\theta$ is a function, which represents the time-cost of travelling (with v_{\max}) along an edge, which connects two vertices in RVG_g . The $t(d_{\text{air}}(w, n))_\theta$ is a function, which represents the time-cost of travelling, with v_{\max} along the airline distance between configurations w and n at angle θ . The list Open contains at any instant, the vertices that are candidates for consideration next. All the vertices of the environment are initially marked as unvisited and useful. Once the algorithm is executed it returns failure if no motion from S to G exists, otherwise a time-minimal motion is returned by backtracking the tree Motion from G to S .

Some attention should be paid to the inner loop of the algorithm, which handles the situation in which a vertex of an obstacle has been visited before. Suppose that after the expansion of a configuration y , a configuration x' has been produced. Further suppose that x' corresponds to the x vertex of an obstacle, which has been visited before at a different place and time instance, say at configuration x'' . If the new way of attaining x provides a less time consuming motion than a previously generated motion from S to x (when x is at x'') then the x' is retained for x and the tree Motion is updated. If x is in Open then it is removed from Open and the new $\bar{t}_f(x)$ is calculated (for x') and then it is reinserted in Open. If it is not in Open then the $\bar{t}_f(x)$ is calculated (for x') and it is inserted in Open for later consideration for expansion.

Since the AGV always moves with its maximum velocity it should never visit the same vertex more than once along time-minimal motion. Therefore, a rule is adopted, in which if the same shaft edge in CT is intersected more than once with the sweeping half-line then the intersection which corresponds to the configuration with smallest \bar{t}_f value is the one that is retained. Note that using this rule the size of the visibility graph is bounded by the total number of obstacles' vertices in the scene. The proof of the algorithm's completeness is similar to that of the A^* and can be found in (Hart *et al*, 1968).

6. TIME-MINIMAL THEOREM

Any motion from S to G , which bends at a location other than an obstacle's vertex (motion x in Figure 5) can be shortcut by a vertex-to-vertex motion from S to G . Since the algorithm presented in this paper is admissible, it is guaranteed that there is no quicker vertex-to-vertex motion between S and G . What has to be demonstrated for the time minimality, is that there is no other arbitrary motion from S to G for the AGV A which can be completed in less time than the one suggested by D*MECHA. An arbitrary motion may be defined in such a way, that the AGV stops at a certain point to wait for an obstacle to move out of its way and then start moving again towards the goal location G .

Lemma 1

The time-minimal motion from S to G , which avoids collisions with the scene's obstacle P , is a vertex-to-vertex motion.

Proof

The proof of the Lemma will be carried out in the two-dimensional configuration space C . Consider the environment of Figure 5, where S and G are the AGV's start and goal locations respectively and the two-dimensional object P is a moving obstacle.

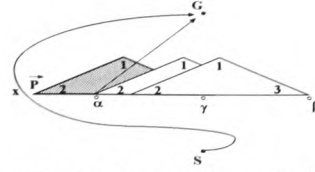


Fig. 5. The goal point is reachable and visible from α .

If A moves straight from S to G it will collide with the obstacle P at location γ at time t_1 , given that A is moving with constant velocity v_{max} . However A can meet vertex 2 and 3 at locations α and β , at times t_2 and t_3 respectively with $t_2, t_3 > t_1$. Even though time $t_2 > t_1$, it is possible to show that a motion from S to G through vertex α is less time consuming than a motion in which the AGV will have to stop and wait in γ . No matter which of the two routes, $\{S, \alpha, G\}$ or $\{S, \gamma, G\}$ the AGV follows, at time t_2 it will be either at position α or γ respectively. When the AGV is at location α at time t_2 , either the goal location G , is visible or not visible. If the goal location G is visible (Figure 5), then the AGV can start moving towards it at time t_2 and reach it. According to the formulation of the problem the velocity of any obstacle in the environment is less than the AGV's velocity. Therefore, by using triangle inequalities over $\{\alpha, \gamma, G\}$, it can be obtained that motion $\{S, \alpha, G\}$ is faster than motion $\{S, \alpha, \gamma, G\}$ hence faster than motion $\{S, \gamma, G\}$ and therefore time-minimal. In the same manner time-minimality holds for one-dimensional obstacles.

If the goal location G is not visible from α at the time instance t_2 then there are two possible outcomes. The first is that G will be visible from α , while the AGV is moving from α to G with constant velocity equal to v_{max} and the obstacle P is moving in the indicated direction (Figure 6) with velocity equal to v_P . Note that the visibility in this case is to be identified in CT. The visibility depends on the angle θ of the sweep line and the slope of the obstacle in CT, this means that the visibility depends on the velocities of the AGV and the obstacle respectively. Even though G is not visible from α at time t_2 it becomes visible as the time passes and the obstacle P moves out of its way and finally A reaches G in less time than the motion $\{S, \gamma, G\}$, for the same reason as in the previous case. In this case, while the AGV is moving from α to G , it might coincide with some points on P 's boundary for some time.

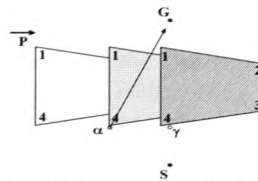


Fig. 6. This figure illustrates the first case.

The second case is when G is not visible from α at time t_2 and it does not become visible even after the

AGV departs from α at time t_2 and while is moving towards G. What happens in this case is that the velocity of the obstacle P is not large enough in comparison to the AGV's velocity in order for P to move out of the AGV's way before a collision occurs between them. In such a case the AGV will have to pass through another vertex of the obstacle before it reaches G. Figure 7a illustrates this case, where it can be seen that if the AGV starts moving from α at time t_2 towards G, with velocity equal to v_{max} , it will eventually collide with the obstacle at the point c.

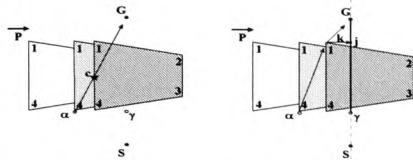


Fig. 7a-b. These figures illustrate the second case.

Figure 7b illustrates the motion of the AGV, in which it will have to pass through vertex 1. For the proof of the Lemma what remains to be shown is that the motion $\{S, \alpha, 1, G\}$ is less time consuming than $\{S, \gamma, G\}$. Since the AGV at time t_2 will be either at position α or γ respectively it has to be shown that the motion from α to G through 1 is faster than the motion from α to G through γ . If A leaves α at time t_2 it will reach vertex 1 at time t_3 at a location on the half plane defined by the line passing through γ and G which contains α , since G is not visible from α in CT at time t_2 . The fact that vertex 1 at t_3 is on the half-plane, which contains α means that G is not visible from γ at t_3 . If j is the projection of 1 on the path of the motion $\{\gamma, G\}$ and if the AGV departs from 1 at t_3 and moves along the straight motion k (with velocity equal to that of the obstacle) it will then join the motion $\{\gamma, G\}$ at point j and time say t_4 . By using triangle inequalities it is derived that the motion from 1 to G is faster than the motion 1 to G through j. Therefore the motion $\{S, \alpha, 1, G\}$ is less time consuming than the motion $\{S, \alpha, 1, j, G\}$ and thus less time consuming than the $\{S, \gamma, G\}$. Note that while the AGV follows the motion $\{\alpha, 1, G\}$, it might coincide with the boundary of P for some time.

The time-minimal theorem

A time-minimal collision-free motion from the start location S to the goal location G, is a vertex-to-vertex motion.

Proof

The theorem is proved by induction on n, the total number of obstacles in the scene. In the base case (i.e., when $n=0$) the theorem holds, since the time-minimal motion between two points is on the straight-line segment which connects them. Assume that the theorem holds for $n-1$ obstacles in the environment. By the induction hypothesis a time-minimal collision-free motion μ exists from S to G for $n-1$ obstacles and this is a vertex-to-vertex motion. To prove the

theorem for n obstacles, an obstacle is inserted in the environment so the total number of obstacles is n. Note that the sequence that the obstacles get involved for the construction of the motion is very important. If the new obstacle collides with motion μ at point k then by the time this collision is avoided the rest of motion μ no longer the same. This means that the sequence that the obstacles get involved from k to G no longer leads to optimum motion and the induction assumption has been violated. Since the environment is time-dependent after the insertion of the n^{th} obstacle if μ is still collision-free the theorem holds. Otherwise the sequence that the obstacles get involved for the construction of the motion is reconsidered. The induction hypothesis is applied for the first $n-1$ (time-wise) obstacles from S to G in the scene, resulting from removing the last obstacle. According to the induction hypothesis this guarantees that there exist a time-minimal collision-free motion μ from S to G. The last obstacle is then reinserted, if motion μ is still collision-free then the theorem holds. Otherwise a time-minimal collision-free motion can be constructed from the last vertex before the collision in motion μ , to G and by Lemma 1 this is a vertex-to-vertex motion. Thus the time-minimal collision-free motion from S to G is the concatenation of the two motions and therefore the theorem holds. The above proof shows that the motion from S to G, obtained by D*MECHA is less time consuming than any other arbitrary motion and is therefore time-minimal.

7. DISCUSSION

It has been demonstrated that D*MECHA algorithm establishes the time-minimal motion from S to G in a two-dimensional time-varying environment. The algorithm is in $O(n^2 \log n)$, where n is the total number of C-Obstacles' vertices. A detailed empirical time analysis of the algorithm can be found in Diamantopoulos *et al*, (1999). If only stationary obstacles populate the environment then the time-minimal motion also defines the shortest path.

8. REFERENCES

- Diamantopoulos, A., G.N. Roberts and D.J. Harwood (1999). A heuristic algorithm for motion planning of an AGV in dynamic environments. *Proc. of the 2nd WESIC*, Newport, UK, 207-214.
- Hart, P.E., N.J. Nilsson and B. Raphael (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions of Systems Science and Cybernetics*, 4 (2), 100-107.
- Hwang, Y.K. and N. Ahuja (1992). Gross Motion Planning-A Survey. *ACM Computing Survey*, 24 (3), 219-291.

EXTENSION TO THE D*MECHA ALGORITHM

A. Diamantopoulos¹, G. N. Roberts¹ and D. J. Harwood²

¹*Mechatronics Research Centre*

²*Engineering Department*

University of Wales College, Newport

Allt-yr-yn Campus, PO Box 180, Newport, NP20 5XR, UK

Tel: ++44 (0) 1633 43248. Fax: ++44 (0) 1633 432442. E-mail: anastasios.diamantopoulos@newport.ac.uk

Keywords: AGV, Motion Planning, Algorithms, and Computational Geometry.

Abstract

In a recent paper, the authors presented an algorithm for finding the time-minimal motion for an AGV (point-like robot) in a two-dimensional environment populated by moving obstacles. In that approach the obstacles were moving along linear paths with constant velocity. The algorithm's asymptotic time was shown to be in $O(n^2 \log n)$, where n is the total number of the obstacles' vertices in the scene. In this paper an investigation for a possible relaxation of some assumptions about the robot's environment is carried out, the applicability of the algorithm is questioned and the possible affect in its computational time is examined. The assumption that the obstacles move along linear paths with constant velocity is relaxed so the obstacles can accomplish piecewise linear motions with constant velocity over each of them.

1 Introduction

Robot motion planning is an extremely broad field in its own right, which has attract much research attention in the last two decades. There are many techniques and algorithms developed for solving the robot motion planning problem under various types of constraints. For good surveys on the robot motion planning problem see [1] and [2], also a good reference, which discuss various motion planning techniques in a fair detail is [3].

In this paper the problem of motion planning for an AGV in time-varying environments is investigated. Since the environment is reconfigured over the time the solution to this problem does not only answers the question 'where is the AGV to go?' but also the question 'when is the AGV to go?' Note that in such environments a motion could be collision-free in a specific period of time and not free of collision in a different period of time.

The specific instance of motion planning in an environment where the obstacles have piecewise linear motion is considered. The applicability of the D*MECHA algorithm [4] is examined and its optimality is explored. Some computational complexity issues are also discussed.

2 Related work

In this section a very brief background literature survey is presented, which is by no means exhaustive. In [5], Sutner and Maass consider the motion planning problem for a point-robot with bounded velocity among dynamic obstacles in one dimension. In their approach they constructed the two-dimensional configuration space-time in which the obstacles were polygonal objects and then they found a collision-free motion in polynomial time on the total number of vertices of the polygonal space-time obstacles.

Kant and Zucker [6], decompose the trajectory planning problem (TPP) into two sub-problems. (i) The path planning problem (PPP) in which a path, which avoids collisions with static obstacles, is planned and (ii) the velocity planning problem (VPP) in which the velocity, which avoids collisions with moving obstacles along this path, is planned. The VPP is posed in path-time space where time is explicitly represented as an extra dimension reducing the problem to a graph search leading to the transformation of the VPP into a PPP. The limitation of this approach is that the AGV is not allowed to alter its path but only its velocity. This means that the algorithm will not find a solution when one exists for the case where an obstacle is moving on the path of the AGV.

In [7], Erdmann and Lozano-Pérez consider the problem of planning motion for multiple robots. They assign priorities to each robot and then they plan the motion of one robot at a time according to its priority. The configuration space-time was represented as a list of configuration space slices at particular points in time. These times are the ones at which some moving object whose motion has already be planned changes its velocity. All paths between adjacent slices, which terminate at obstacles' vertices, are considered for the construction of the motion. A motion was then obtained by using vertex-to-vertex translation of the point-robot in between the configuration slices.

Fujimura and Samet in [8] presented an algorithm to find a motion for a point-robot, in an environment populated by time-dependent obstacles and destination point. Their approach is based on the accessibility concept and produces the time-minimal motion between the robot's start and goal (which is moving) points in polynomial time on the total number of the obstacle vertices, given that the robot has larger velocity than the obstacles.

Canny and Reif, in [9] show that motion planning for a point-robot, with a bounded velocity modulus, in a two-dimensional environment populated by arbitrarily many

moving, non-rotating convex obstacles, that move at constant velocity is NP-hard.

In [10] it was shown that the problem of motion planning in a three-dimensional environment populated by moving obstacles is a PSPACE-hard problem, when the robot's velocity modulus is bounded and NP-hard when the robot's velocity modulus is not bounded.

3 The D*MECHA Algorithm

The authors in [4] presented an approach for finding the time-minimal motion for a point robot between two points in a two-dimensional environment populated by static and moving obstacles. The environment's time-dependent obstacles are moving along linear paths with constant velocity less than the velocity of the AGV. This approach constructs the robot's configuration space-time and then searches it for the shortest path using heuristics by taking into consideration the path's monotonicity constraint and the maximum velocity constraint. Figure 1 illustrates the AGV's two-dimensional configuration space C and its three-dimensional configuration space-time CT .

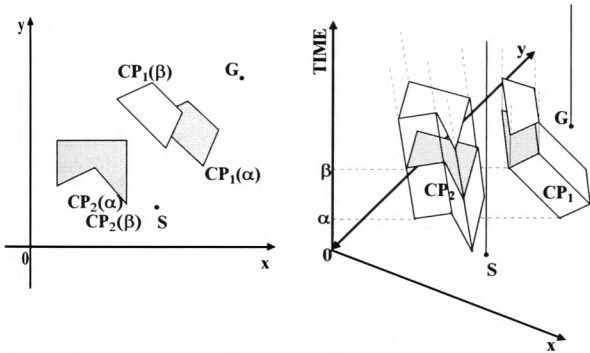


Figure 1. This figure illustrates the AGV's configuration space C and configuration space-time CT .

As it can be seen from figure 1 the configuration space-time is a static environment representing the constraints imposed on the AGV by its time-dependent workspace. Once the configuration space-time has been constructed it can then be searched for a path between the start point S and the goal point G . S and G in CT correspond to half-lines emanating from S and G respectively, which are parallel to the time axis.

All the reachable configurations in CT form a single configuration p , are defined by the surface of a cone CN emanating from p . The cone CN is a right circular cone and its slant edge creates an angle ϑ with the xy -plane which is equal to $\arctan(v^{-1})$, where v is the AGV's constant velocity. Therefore if CT is polar-swept with a half-line emanating from a point p , at angle ϑ (with the x - y plane), then all the intersections between the half-line and any of the shaft edges of the prisms correspond to reachable configurations from p .

In order for the AGV to be able to move from configuration p to the reachable configuration, say q , the two configurations should be visible to each other. That is they can get connected with an edge and this edge does not overlap the interior of any obstacle (prism in CT).

The algorithm starts from configuration S at time t_0 and generates a graph, which connects S to G with edges in CT , this graph is a topological representation of the connectivity of the environment. It is a directed graph since the path of the AGV should be strictly monotone in time, which demonstrates the reachability and visibility at angle ϑ and is denoted by RVG_ϑ . The time-minimal motion for the AGV in C corresponds to the shortest path in CT , given that the AGV always moves at its maximum velocity v_{max} . In [4] it was shown that in the time-minimal motion the AGV does not visit acute vertices and extreme vertices of visible sequences. The algorithm was stated as follows.

begin

$\vartheta := \arctan(v_{max}^{-1})$

put S **in** Motion;

put S **in** Open;

mark S visited;

$\hat{t}_g(S) := 0$;

while (Open \neq nil) **do**

begin

$w := \{i \in \text{Open} : \hat{t}_f(i) < \hat{t}_f(j) \mid \forall j \in \text{Open},$

resolve ties arbitrarily but always favour $G\}$;

remove w **from** Open;

if $w = G$ **then** **exit while loop**;

$VV := \{w\text{-visible points at angle } \vartheta\}$;

$EV := \{\text{extreme vertices of the } w\text{-visible sequences in the projections of } VV \text{ into } Q\}$;

mark all the non-extremes vertices useless

$AV := EV - \{\text{obtuse vertices in } EV\}$;

for each vertex $i \in AV$ **do**

if i **is not marked useless then**

if i **is not marked visited then**

begin

$\hat{t}_h(i) := t(d_{air}(i, G))_\vartheta$ (the time airline distance from i to G);

$\hat{t}_g(i) := \hat{t}_g(w) + t(d(w, i))_\vartheta$;

$\hat{t}_f(i) := \hat{t}_g(i) + \hat{t}_h(i)$;

put i **in** Motion **with pointer toward** w ;

put i **in** Open;

mark i visited;

end;

else if $\hat{t}_f(i) > \hat{t}_g(w) + t(d(w, i))_\vartheta +$

$t(d_{air}(i, G))_\vartheta$ **then**

begin

redirect pointer of i **toward** w **in** Motion;

if $i \in \text{Open}$ **then** **remove** i **from** Open

$\hat{t}_g(i) := \hat{t}_g(w) + t(d(w, i))_\vartheta$;

$\hat{t}_f(i) := \hat{t}_g(i) + \hat{t}_h(i)$;

put i **in** Open;

end;

end;

if (Open \neq nil) **then** **return** Motion **by tracing all the pointers backward from** G **to** S

else return failure;

end.

This algorithm uses the heuristics of the A* algorithm and finds the time-minimal motion for an AGV (point-robot) from S to G in an environment populated by two-dimensional static and moving obstacles. The proofs of the algorithm's admissibility and completeness is similar to that of the A* and are discussed in [11]. In [4] it was proved that the time-minimal motion in environments like the aforementioned, is a vertex-to-vertex motion and since the D*MECHA algorithm is admissible it is guaranteed to find the time-minimal motion in general from S to G.

4 Piecewise linear motion

In this section the applicability of the D*MECHA algorithm in time-varying environments, in which the obstacles have piecewise linear motion, is explored. This environment is more general in a sense that is more realistic than the one in which the obstacles have linear motion. In the piecewise linear motion the obstacles move with constant velocity in a fixed direction for a finite number of time intervals. Their velocity in different time intervals does not have to be the same. Note that when an obstacle changes its direction and velocity within a time interval the direction of the slope of the corresponding prism in CT changes. Figure 2 illustrates the AGV's two-dimensional configuration space C and its three-dimensional configuration space-time CT.

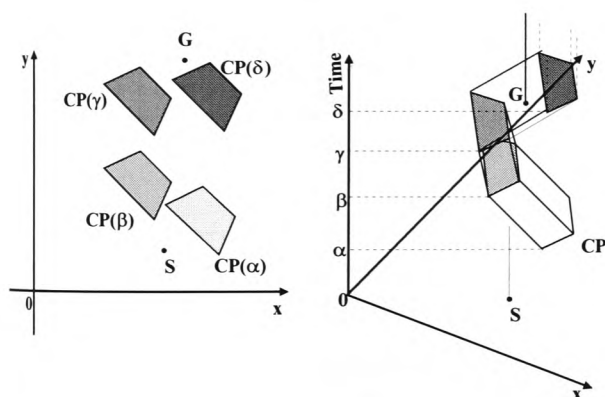


Figure 2. This figure illustrates the AGV's configuration space C and configuration space-time CT, of an environment, which contains an obstacle with piecewise linear motion.

In [4] it was stated that the computational complexity of D*MECHA is $O(n^2 \log n)$ for an environment populated by static and moving obstacles along linear paths, where n is the total number of the C-Obstacles' vertices. However it has to be determined whether the overall time complexity changes when the obstacles have piecewise linear motion. Suppose that the direction and velocity of an obstacle P changes m times, a vertex v of P is reachable and visible from a given point only once, given that the AGV moves with its maximum velocity constantly. Therefore by the above observation and by keeping in mind that the process is the same it is concluded that the overall worst-case complexity of the D*MECHA remains unchanged. Summarising the above concepts it is concluded that D*MECHA is in $O(n^2 \log n)$ even when the moving obstacles of the environment have

piecewise linear motion, where n is the total number of the obstacles' vertices. This time improves the current asymptotic time $O(n^2 \log(nm))$ given in [12], where n is the total number of the obstacles vertices and m is the average number of turns made by the obstacles.

Once the applicability of D*MECHA for such environments has been decided what is left to be investigated is whether the motion obtained is still time minimal.

5 Time-minimality of the motion

In this section the time-minimality of the D*MECHA algorithm, when applied in environments with static and time-varying obstacles with piecewise linear motion, is examined. In [4] it was shown that the time-minimal motion from S to G in an environment which contains static and moving along linear paths obstacles, is a vertex-to-vertex motion. Figure 3 illustrates an example, which suggests that this is not the case when the obstacles have piecewise linear motion. Suppose that the obstacle P starts moving from position ℓ_1 towards the bottom with velocity 2m/sec. When the edge (1, 2) reaches the line ℓ_2 the direction and the velocity of the obstacle change towards the indicated direction with velocity 1m/sec.

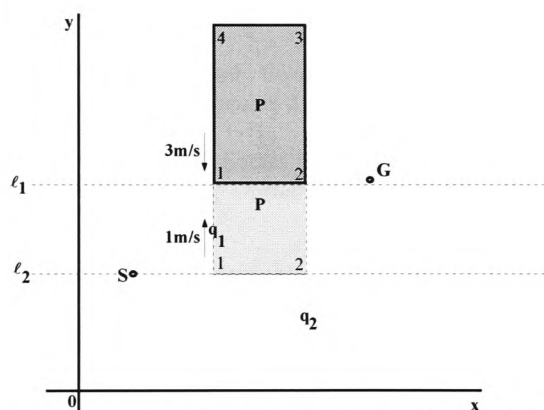


Figure 3. An example of an obstacle in piecewise linear motion.

Suppose that the time-minimal motion is the one, which goes around the bottom side of the obstacle. Further suppose that given the speed limit imposed on the AGV, the AGV leaves the start point S at time t_0 it can reach the vertex 1 at a configuration q_1 at time t_1 or vertex 2 at configuration q_2 and time t_2 , if the obstacle would not have changed its direction. However the obstacle changes its direction and its velocity when its edge (1, 2) reaches the line ℓ_2 . This means that if the AGV starts moving from S at time t_0 it will never reach vertex 2 at q_2 simply because the obstacle will never be at such a configuration. Therefore the motion will have to go through the vertex 1. If the AGV leaves from configuration q_1 at time t_1 and reaches the vertex 2 before the obstacle changes its direction at a configuration q_3 and time t_3 then it can start moving from q_3 and reach the goal point and this motion is time-minimal. Otherwise the AGV will have to move through the other vertices of the obstacle and therefore this motion will not be the time-minimal motion.

When the motion of the obstacles is piecewise linear the direction of the obstacles changes over time. Therefore it is possible for the AGV not to reach a vertex of the obstacle due to the fact that the obstacle changed its direction. For example in figure 3 vertex 1 is visible and reachable from S but vertex 2 it is not, due to the fact that the obstacle's direction changes before the AGV meets vertex 2 in the configuration q_2 . However, there is an internal point on the edge, which is adjacent to these vertices and can serve as a configuration for the AGV to pass through on its way to the goal position, this point is called assistant point. Therefore the motion from S to G produced by the algorithm is a vertex-to-vertex motion, which goes through vertices of the obstacles and assistant configurations. The assistant configurations correspond to reachable and visible configuration from a configuration q in CT on a common edge between two prisms. This configuration is an intersection of the sweep-line and such an edge in CT. In the example of figure 4, q_{as} is the assistant point from S on the edge (1, 2). The time minimal motion from S to G is $\{S, q_{as}, q_3, G\}$, figure 4 illustrates the motion. Note that in this motion the AGV coincides with the boundary of the obstacle for some time.

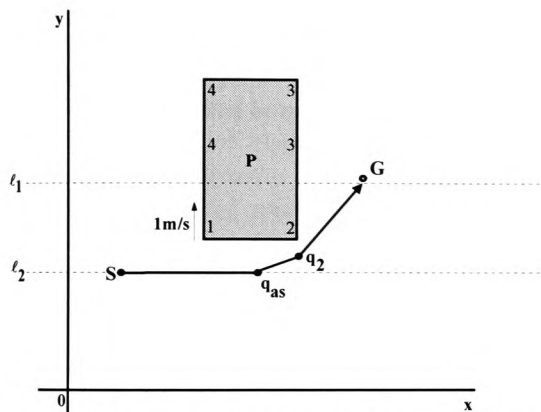


Figure 4. This figure illustrates the time-minimal motion from S to G passing through the assistant configuration q_{as} .

In [4] the time-minimality theorem states that the time-minimal motion from S to G in an environment populated by linear moving obstacles is a vertex-to-vertex motion. This theorem also holds for environments, which contain piecewise linearly moving obstacles. The only change is that in such environments the assistant points considered as vertices.

6 Discussion

In this paper the applicability of the algorithm proposed by the authors in [4] in environments which contain piecewise linearly moving obstacles was investigated. The description of this environment is somewhat more general than the linear motion of the obstacles. The D*MECHA algorithm is applicable in such environments and finds the time-minimal motion between two points in an environment populated by static, linearly moving and piecewise linearly moving obstacles in $O(n^2 \log n)$ time, where n is the total number of the obstacles' vertices.

References

- [1] Hwang, Y.K. and Ahuja N., Gross Motion Planning - A Survey. *ACM Computing Survey*, **24** (3), 219-291, (1992).
- [2] Schwartz J. T. and Sharir M., A Survey of Motion planning and Related Geometric Algorithms. *Artificial Intelligence*, **37**, 157-169, (1988).
- [3] Latombe J. C., Robot motion planning, *Kluwer Academic Publishers*, (1991).
- [4] Diamantopoulos A., Roberts G. N. and Harwood D. J., An improved algorithm for finding the time-minimal motion for an AGV in time-dependent environments, *To be published in the Proceeding of 6th IFAC SYMPOSIUM on ROBOT CONTROL, SYROCO 2000*, September 21-23 2000, Vienna, Austria, (2000).
- [5] Sutner M. and Maass W., Motion planning among time dependent obstacles, *Acta Informatica*, **26**, 93-122, (1988).
- [6] Kant K. and Zucker S. W., Toward Efficient Trajectory Planning: The Path-Velocity Decomposition, *The International Journal of Robotics Research*, **5** (3), 72-89, (1986).
- [7] Erdmann M. and Lozano-Pérez T., On multiple moving objects, *Algorithmica*, **2** (4), 477-522, (1987).
- [8] Fujimura K. and Samet H., Planning a Time-Minimal Motion Among Moving Obstacles, *Algorithmica*, **10**, 41-63, (1993).
- [9] Canny J. and Reif J., New Lower Bound Techniques for Robot Motion Planning Problems, *IEEE 28th Annual Symposium on Foundations of Computer Science*, 49-60, (1987).
- [10] Reif J. and Sharir M., Motion Planning in the Presence of Moving Obstacles, *IEEE 26th Annual Symposium on Foundations of Computer Science*, 144-154, (1985).
- [11] Hart P. E., Nilsson N. J. and Raphael B., A Formal Basis for the Heuristic Determination of Minimum Cost paths, *IEEE Transaction on Systems, Science and Cybernetics*, **4** (2), 100-107, (1968).
- [12] Fujimura K., Motion planning in Dynamic Environments, Springer - Verlag, Tokyo, (1991).

Robot Motion Planning in Environments Populated by Shrinking and Expanding Obstacles

A. Diamantopoulos¹, G. N. Roberts¹ and D. J. Harwood²

¹Mechatronics Research Centre

²Engineering Department

University of Wales College, Newport

Allt-yr-yn Campus. PO Box 180,

Newport, NP20 5XR

United Kingdom

Tel: +44 (0) 1633 432487, FAX: +44 (0) 1633 432442.

e-mail: anastasios.diamantopoulos@newport.ac.uk

Abstract - This paper address the robot motion planning problem in an environment populated by obstacles that change their size over time. That is the obstacles in the environment can shrink or expand over time. This is a slightly modified problem to the classic dynamic robot motion planning problem. In the classic problem the position of the obstacles in the environment is time dependent whereas in this instance the size of the obstacles in the environment depends on time. Intuitively it seems that there is not large application domain or at least frequent applications of the real life, which can be modelled using the concept of shrinking and expanding obstacles. However in this paper some applications are discussed and the problem of moving an AGV in such environments is considered. More specifically an algorithm for finding the time-minimal motion between two points in an environment populated by shrinking and expanding obstacles is considered.

Keywords: Motion Planning, Path Planning, Obstacle Avoidance, Time-minimal Motion, and Computational Geometry.

1. Introduction

Many researchers have considered the robot motion planning problem in dynamic environments. Sutner and Maass in [1] have

solved the problem of motion planning in a dynamic environment populated by one-dimensional time varying obstacles. They constructed the robot's space-time configuration space and they solve the problem in polynomial time on the total number of the vertices of the polygonal space-time obstacles.

In [2], it has been shown that motion planning for a point-robot, with a bounded velocity modulus, in a two-dimensional environment populated by arbitrarily many moving, non-rotating convex obstacles, that move at constant velocity is NP-hard. Erdmann and Lozano-Pérez [3] consider the problem of planning motion for multiple robots. They assign priorities to each robot and then they plan the motion of one robot at a time according to its priority. Kant and Zucker [4] consider a monotonous path in space-time by decomposing the trajectory planning problem into two sub-problems: (i) planning a path to avoid collisions with static obstacles and (ii) planning the velocity along the path to avoid collisions with moving obstacles. In [5] an algorithm was presented to find a motion for a point-robot, in an environment populated by time-depended obstacles and destination point. The proposed algorithm finds the time-minimal motion given that the point-robot moves faster than the obstacles and the destination point, in polynomial time using the accessibility concept.

In [6], an algorithm was proposed for finding a greedy in time motion for an AGV in an environment populated by linear moving obstacles. The algorithm uses the concept of the visibility graph to find a motion in the time-space. The AGV was a point robot subjected only to speed upper bound and the algorithm had a polynomial computational time on the total number of the obstacles' vertices. Fujimura in [7] proposed two algorithms for solving motion planning problem for a point robot that moves in an environment where the moving obstacles and the destination point have cyclic motion. These algorithms are the *hit-and-leave*, which is suited for sensor-based navigation and the *accessibility* algorithm, which is more suited when the environment is accurately known ahead planning. Both of these algorithms establish a collision-free motion, providing that the robot moves faster than the obstacles. The motion defined by the second method is also time-minimal.

Reif and Sharir in [8], showed that the problem of motion planning in a three-dimensional environment populated by moving obstacles is a PSPACE-hard problem, when the robot's velocity modulus is bounded and NP-hard when the robot's velocity modulus is not bounded.

Fujimura has considered the problem of motion planning among shrinking and expanded obstacles in [9]. His algorithm was based on a wave propagation technique and it had polynomial computational time on the total number of the environment's vertices. The approach presented in this paper finds the time-minimal motion for a point robot between two points, in an environment populated by shrinking and expanding obstacles, in computational time $O(n^2 \log n)$, where n is the total number of the obstacles' vertices. This approach is based on the D*MECHA algorithm previously reported by the authors [6].

2. Motivation

A real world application which can modelled as an environment containing shrinking and expanding obstacles is when a robot watercraft is navigating in a sea in which tide give rise into a difference of the water's level on skerries and islands. The immediate affect on the robot's workspace is the shrink or

expansion of the environment's obstacles. Obstacles are considered the cross sections of the islands with the plane defined by the sea level. Another application is when an AGV operates in a manufacturing environment where there are round multi-arms stations that operate as part of the manufacturing process. As these stations extend and retract their arms over the time they can be considered as shrinking and expanding obstacles for the AGV. Another possible application is when an AGV is moving in dynamic environments and there are uncertainties on the obstacles' velocity, they can be modelled as moving and expanding obstacles in this way the collisions due to the uncertainty can be avoided.

3. Establishment of the problem

Consider the environment W populated by shrinking and expanding obstacles P_i , where $i \in \mathbb{N}$ and the AGV's initial point S and goal point G . The problem is to establish a time-minimal motion from S to G for the AGV A avoiding collisions with the environment's obstacles, providing there exist one. The AGV A is a point robot, which moves with constant velocity. The description of the obstacles (such as shapes, locations and velocities) is accurately known ahead of planning.

3.1. Description of the obstacles

Shrinking and expanding obstacles are defined as two-dimensional convex polytopes whose boundary alters over the time. The alteration does not happen randomly but in the following manner; every vertex V_i of an obstacle is moving with constant velocity along the linear path defined by a fixed point O inside the obstacle and the vertex V_i .

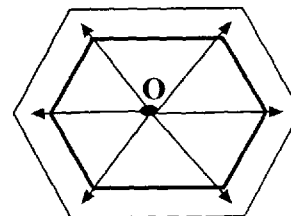


Figure 1. This figure illustrates a growing obstacle.

The distance that an obstacle's vertex can cover within time t is given by $c \cdot t \cdot v_{P_i}$,

where c is a constant which can only take two values, 1 and -1 depending whether the obstacle is expanding or shrinking and v_{pi} is the speed of the vertices of the i obstacle. Note that all the vertices of an obstacle start moving and finishing together (this assumption may be relaxed). The obstacles' vertices move in such a way that the obstacles do not deform. Every dynamic obstacle, before the beginning and after the end of its motion has velocity equal to zero. The environment's obstacles are not allowed to collide at any time. Any contact between the obstacle's boundaries is considered a collision.

4. Construction of the Space-Time Configuration Space

Since the environment is dynamically reconfigured for the solution of the path planning problem the parameter of time has to be taken into consideration. Note that a path π could be collision-free at a specific time instance t_1 and not free of collisions in a different time instance t_2 . Therefore in such environments the path from the AGV's initial point to its goal point has to be defined as a function of time and the term motion is used instead of path, hence motion planning instead of path planning.

It is possible to construct the AGV's configuration space at any fixed-point t in time, this geometrically captures the constraints on the AGV's degrees of freedom at time t . Now considering all points in time, the space-time configuration space can be produced. The space-time configuration space is defined by adding the dimension of time into the configuration space C . By the formulation of the problem the AGV is a point-robot therefore its configuration space $C = W = R^2$ and every obstacle's configuration space $CP_i = P_i$. The space-time configuration space $CT = R^2 \times [0, +\infty)$ and every obstacle's configuration space CP_i map from C into CT to a prism denoted by CTP_i . Figures 1a and 1b illustrate the AGV's configuration space C and space-time configuration space CT . Since the boundaries of the CP_i do not come in contact at any time the boundaries of the CTP_i do not come in contact at any time.

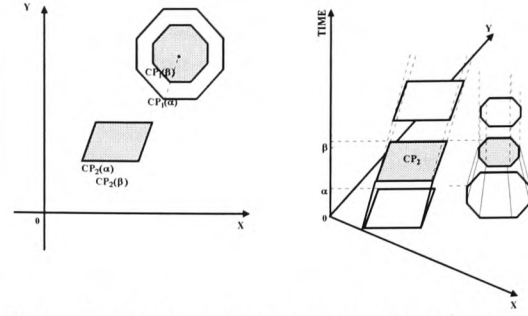


Figure 2. This figure illustrates the configuration space C and the space-time configuration space CT .

From the figure 2 it can be noticed that the edges of the CTP_i in CT that do not constitute the bases of the prisms (shaft edges) correspond to vertices of the CP_i in C . The edges of the CTP_i that constitute the prisms' bases in CT correspond to the edges of CP_i in C . The AGV's start point S and its goal point G form C map into CT to half-lines emanating from S and G respectively parallel to the time axis. A slice of the space-time configuration space CT at a given time t corresponds to the AGV's configuration space at time t . Once the space-time configuration space has been constructed it can then be searched for the time-minimal motion between the initial and goal positions. Erdman and Lozano-Pérez [3], used the idea of space-time configuration space in order to plan motions for multiple robots. They assign priorities to each robot and then they plan the motion of one robot at a time according to its priority. The configuration space-time was constructed for each moving object in order to represent the constraints imposed on it by its time-depended environment. In their approach the configuration space-time was discretised and represented as a list of configuration space slices. A motion was then obtained by using vertex-to-vertex translation of the point-robot between adjacent slices. This planner is time resolution-complete between the slices, unless the free space-time between slices is also searched.

In this paper the space-time configuration space is constructed in a continuous manner and then searched for a motion between the AGV's initial and goal positions. In the next section a method to search the CT will be presented.

5. Searching the CT for a Collision-free motion

In the time-minimal motion the AGV A has to move at its maximum velocity (v_{\max}). Therefore, given that the AGV A is moving with constant velocity equal to v_{\max} , the set of all the configurations that can reach from a configuration p in space-time between time t_1 and t_2 , is defined by the surface of the cone CN which emanates from p.

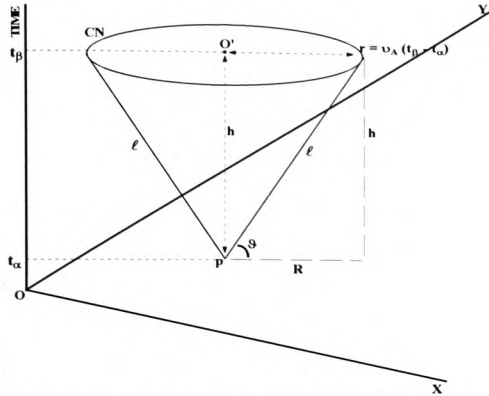


Figure 3. The surface of the cone CN defines the set of all reachable configurations from p.

The cone CN is a right cone which emanates from p. The height h of the cone is equal to $t_2 - t_1$. The radius of the base of CN corresponds to the distance that the AGV can travel in the x-y plane in time $t_2 - t_1$ given that it is moving with velocity v_A equals to v_{\max} and is defined as $r = (t_2 - t_1) v_A$. The angle ϑ created by the slant height ℓ of the cone and the x-y plane is defined as follows:

$$\begin{aligned} \tan \vartheta &= \frac{h}{R} \Leftrightarrow \tan \vartheta = \frac{t_\beta - t_\alpha}{v(t_\beta - t_\alpha)} \Leftrightarrow \\ \tan \vartheta &= \frac{1}{v} \Leftrightarrow \tan \vartheta = v^{-1} \Rightarrow \\ \vartheta &= \tan^{-1}(v^{-1}) \end{aligned}$$

So if the environment is polar-swept with a ray r emanating from a configuration say p then all the intersections between the ray r and any shaft edge of the CTP_i , define the set of reachable configurations from p. Now consider the reachable from p configuration q and its projections into the x-y plane and the time axis x_q , y_q and t_q respectively. These projections

correspond to the position and the time instance that the AGV and the corresponding C-Obstacle's vertex can meet in C, given that the AGV is moving with constant velocity equal to v_{\max} .

However, in order to say that an AGV is capable of moving from configuration p to configuration q, 'reachability' is not enough. Another condition, which has to be satisfied, is visibility. This means that the two configurations can be connected with an edge and this edge does not overlap the interior of a CTP_i in CT. When the configuration p and the configuration q can be connected with an edge, it is said that q is visible from p. In summary, if a ray v is swept about p by keeping a constant angle $\vartheta = \tan^{-1}(v_A^{-1})$ with the x-y plane, all the p-visible configurations at angle ϑ in the CT can be identified. These p-visible configurations at angle ϑ in CT correspond to p-visible vertices in C at a specific location at a specific time instance, which can be reached from p by the AGV, given that is moving with constant velocity equal to v_A . In this way a graph demonstrating 'reachability' and visibility ('reacha-visibility graph') at angle ϑ (RVG_ϑ) can be constructed in CT and then searched for a path. Note that no object can move back in time so the motion established by the algorithm has to be strictly monotone in time and therefore the RVG_ϑ is a directed graph. The following proposition is used in order to reduce the size of the RVG_ϑ .

Proposition 1

A time-minimal motion never visits configurations in CT, which correspond to non-extreme vertices of a visible sequence¹ in C.

Proposition 1 is applied to the two-dimensional projection of the CTP_i in the x-y plane. This proposition was proved by the authors in [6].

6. The proposed algorithm

The algorithm presented in this paper is an variant of the D*MECHA algorithm previously reported by the authors in [6]. The proposed algorithm is a heuristic search

¹ Visible sequence is a set, which contains all visible vertices from a given vertex, which are consecutive on a single obstacle's boundary.

strategy, based on the A* algorithm [10]. The algorithm searches for the Euclidean shortest path between the start and goal half-lines in CT. This path (shortest) corresponds in the time-minimal motion from S to G in C.

The algorithm is a heuristic search process guided by the evaluation of the function \hat{t}_f over every configuration considered for the construction of the motion. An estimation function is used to assess how close a current configuration c is to the goal configuration. If there is more than one configuration for the robot to move toward, a heuristic function is applied over each of them in order to choose the one with the best assessment. The remaining configurations are stored in a candidate's list for later consideration.

Since the AGV is moving with constant velocity equal to v_{\max} , then the shortest Euclidean path in CT corresponds to the time-minimal motion in C. Therefore the airline distance between the current configuration c and the goal half-line can be chosen as a heuristic function. Note that this distance to the goal half-line should be measured from the current configuration c to a configuration o on the goal half-line in such a way that the line-segment co creates an angle ϑ with the x-y plane. The length of this line-segment is the smallest possible distance between the current configuration and the goal half-line in CT, which corresponds to the time-minimal motion in C. The search in CT can be carried out either based on information about the Euclidean distance or based on information about the travel time (distance over speed). In this approach information about the travel time, has been chosen for the search of CT.

The function $\hat{t}_f(n)$ at any configuration n is an estimation of $t_f(n)$ which defines the time-cost of travelling along the path traced by the actual time-minimal motion from S to G constrained to pass through configuration n . More formally $\hat{t}_f(n) = \hat{t}_g(n) + \hat{t}_h(n)$, where $\hat{t}_g(n)$ is the time-cost of travelling along the path traced by the time-minimal motion from S to n the algorithm found so far, estimating $t_g(n)$, which defines the time-cost of travelling along the path traced by the actual time-minimal motion from S to n . The function $\hat{t}_h(n)$ is an estimation of the function $t_h(n)$ which defines

the time-cost of travelling along the path traced by the actual time-minimal motion from configuration n to a preferred goal of n (in this case G). However an estimation of the function $t_h(n)$ is not easy to find, so the best way to define it is to rely on information about the problem domain. In time-minimal applications when the AGV travels with constant speed, a good estimation of $t_h(n)$ is $\hat{t}_h(n)$, the travel time of along the airline distance from n to the goal G. This distance is the smallest possible distance between these configurations, so $\hat{t}_h(n)$ is the lower bound of $t_h(n)$. When $\hat{h}(n)$ is the lower bound the algorithm is admissible, [10]. The proposed algorithm finds the time-minimal motion for an AGV in a dynamic environment and is stated as follows:

The proposed algorithm

```

begin
   $\vartheta := \arctan(v_{\max}^{-1})$ 
  put S in Motion;
  put S in Open;
  mark S visited;
   $\hat{t}_g(S) := 0$ ;
  while (Open  $\neq$  nil) do
    begin
       $w := \{i \in \text{Open} : \hat{t}_f(i) < \hat{t}_f(j) \mid \forall j \in \text{Open},$ 
        resolve ties arbitrarily but always
        in favour of the goal node};
      remove  $w$  from Open;
      if  $w = G$  then exit while loop;
       $VV := \{w\text{-visible points at angle } \vartheta\}$ ;
       $EV := \{\text{extreme vertices of the } w\text{-visible}$ 
        sequences in the projections of  $VV$ 
        into  $Q\}$ ;
      mark all the non-extremes vertices
        useless;
      for each vertex  $i \in EV$  do
        if  $i$  is not marked useless then
          if  $i$  is not marked visited then
            begin
               $\hat{t}_h(i) := t(d_{\text{air}}(i, G))_{\vartheta}$  (the time
                of travelling on the airline
                distance from  $i$  to  $G$ );
               $\hat{t}_g(i) := \hat{t}_g(w) + t(d(w, i))_{\vartheta}$ ;
               $\hat{t}_f(i) := \hat{t}_g(i) + \hat{t}_h(i)$ ;
              put  $i$  in Motion with pointer
                toward  $w$ ;
              put  $i$  in Open;
            end
    end

```

```

    mark i visited;
end;
else if  $\hat{t}_f(i) > \hat{t}_g(w) + t(d(w, i))_9 +$ 
       $t(d_{air}(i, G))_9$  then
begin
  redirect pointer of i toward w in
  Motion;
  if  $i \in \text{Open}$  then remove i from
  Open
   $\hat{t}_g(i) := \hat{t}_g(w) + t(d(w, i))_9;$ 
   $\hat{t}_f(i) := \hat{t}_g(i) + \hat{t}_h(i);$ 
  put i in Open;
end;
end;
if ( $\text{Open} \neq \text{nil}$ ) then return Motion by
  tracing all the pointers
  backward from G to
else return failure;
end.

```

In the above algorithm, Motion is a spanning tree, which represents at any instant the best motion obtained so far. For each visited configuration n (except S) a pointer to its parent is held. The function $\hat{t}_f(n)$ is associated with each configuration n in the current Motion. The $t(d(w, n))_9$ is a function, which represents the time-cost of travelling along an edge, which connects two vertices in RVG_9 , with v_{\max} . The $d_{air}(w, n)_9$ is a function, which represents the time-cost of travelling, with v_{\max} along the airline distance between configurations w and n at angle 9 . The list Open contains at any instant, the vertices that are candidates for consideration next. All the vertices of the environment are initially marked as unvisited and useful. Once the algorithm is executed it returns failure if no motion from S to G exists, otherwise a time-minimal motion is returned by backtracking the spanning tree Motion from G to S .

Some attention should be paid on the inner loop of the algorithm, which handles the situation in which a vertex of an obstacle has been visited before. Suppose that after the expansion of a configuration y a configuration x' has been produced. Further suppose that x' corresponds to the x vertex of an obstacle, which has been visited before at a different place and time instance, say at configuration x'' . If the new way of attaining x provides a

less time consuming motion than a previously generated motion from S to x (when x is at x') then the x' is retained for x and the tree Motion is updated. If x is in Open then it is removed from Open, the new $\hat{t}_f(x)$ is calculated (for x') and then it is reinserted in Open. If it is not in Open then the $\hat{t}_f(x)$ is calculated (for x') and it is inserted in Open for later consideration for

7. Admissibility and Optimality of the algorithm

The algorithm proposed in this paper is based on the A^* algorithm for finding the shortest path in a graph. The RVG_9 is a directed δ graph and the evaluation function used is $\hat{t}_f(n) = \hat{t}_g(n) + \hat{t}_h(n)$, since the $\hat{t}_h(n) \leq t_h(n) \forall n$, then the proposed algorithm is admissible. That is it always terminating by finding an optimal path from S to a preferred node of S in CT and therefore an optimal motion in C. The proof of the algorithm's admissibility is the same with the A^* 's proof and is presented in [10]. In a similar way to the A^* in [10] it can be shown that the proposed algorithm is optimal in a sense that it expands the smallest number of configurations necessary to guarantee finding an optimal motion.

8. Analysis of the algorithm

By analyzing the algorithm it can be noticed that the steps that find n with the smallest \hat{t}_f in Open and remove it from it require $O(n)$ time since there are at most n vertices in Open, the treatment of its 'children' requires also $O(n)$ time. The step for computing the set of visible vertices for a κ vertex at angle 9 with the x - y plane requires $O(n \log n)$ time, (polar sweep algorithm). The computation time for finding the extreme vertices in a κ -visible sequence is $O(n)$, [11]. Since all these steps appear nested in the while loop of the algorithm, which can be repeated at most n times, the overall time complexity of the algorithm is $O(n^2 \log n)$.

9. The time-minimality of the algorithm

In this section it is shown that the motion that the algorithm establishes for an AGV between its initial and goal its points in an environment populated by shrinking and expanding obstacles is time-minimal. This is

achieved by demonstrating that there is no other motion from S to G for the AGV A that can be executed in less time than the one suggested by the proposed algorithm.

Lemma 1

The proposed algorithm establishes the quickest vertex-to-vertex motion from S to G .

Proof

Since the algorithm is admissible it is guaranteed to return the quickest vertex-to-vertex motion.

It is essential for the time-minimality of the algorithm to show that the time-minimal motion is vertex to-vertex motion. Therefore it has to be shown that no other arbitrary motion is faster. An arbitrary motion can be defined in such a way that the AGV will have to stop at a certain point wait for an obstacle to shrink out of its way and the carry on moving towards its goal position.

Theorem 1

The time minimal motion for an AGV A from its start location S to its goal location G , in an environment populated by shrinking and expanding obstacles is a vertex-to-vertex motion.

Proof

The proof of this theorem is carried out in the two-dimensional C . Consider the configuration space C of figure 4.

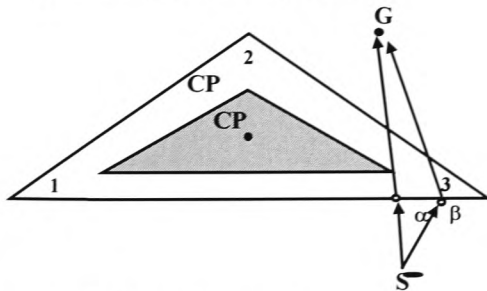


Figure 4. This figure illustrates theorem 1.

The time-minimal motion from S to G is the straight-line motion that connects them. However this motion is not collision-free because if the AGV follows it, it will collide with the CP at location α in time t_1 , given that A is moving with constant velocity equals to v_{max} . However the AGV A can meet vertex 5 of the shrinking CP at location β in time t_2 , with

$t_2 > t_1$. Even though time $t_2 > t_1$, it is possible to show that motion $S\beta G$ is less time consuming than $S\alpha G$. No matter which of the two routes, $S\alpha G$ or $S\beta G$ the AGV follows, in time t_2 it will be either at position α or β respectively. According to the formulation of the problem the velocity of any obstacle in the environment is less than the AGV's velocity. Therefore, by using triangle inequalities over $\beta\alpha G$, it can be obtained that motion $S\beta G$ is faster than motion $S\beta\alpha G$ and hence motion $S\beta G$ is faster than motion $S\alpha G$ and therefore time-minimal.

The above proof shows that the motion from S to G , obtained by the algorithm presented in this paper is less time consuming than any other arbitrarily motion and is therefore time-minimal.

10. Discussion

In this paper the D*MECHA algorithm [6], was reconsidered by the authors and slightly modified to be applicable in environments populated by convex shrinking and expanding obstacles. Its admissibility and time-minimality was shown. Intuitively it is possible for the algorithm to be applicable in environments, which contain both moving, and shrinking/expanding obstacles and further with possible extensions, in environments populated by moving obstacles which shrink and expand. Note that when the obstacles in the environment have velocity equal to zero the time-minimal motion from S to G obtained by the algorithm also defines the shortest path from S to G .

11. References

- [1] Sutner M. and Maass W., 1988, *Motion planning among time dependent obstacles*, Acta Informatica, Vol. 26, pp. 93-122.
- [2] Canny J. and Reif J., 1987, New Lower Bound Techniques for Robot Motion Planning Problems, IEEE 28th Annual Symposium on Foundations of Computer Science, pp. 49-60.
- [3] Erdmann M. and Lozano-Pérez T., 1987, *On multiple moving objects*, Algorithmica, Vol. 2, No. 4, pp. 477-522.

- [4] Kant K. and Zucker S. W., 1986, *Toward Efficient Trajectory Planning: The Path-Velocity Decomposition*, The International Journal of Robotics Research, Vol. 5, No.3, pp 72-89.
- [5] Fujimura K. and Samet H., 1993, *Planning a Time-Minimal Motion Among Moving Obstacles*, Algorithmica, Vol. 10, pp 41-63.
- [6] Diamantopoulos, A., Roberts, G. N. and Harwood, D. J. *A heuristic algorithm for motion planning of an AGV in dynamic environments*. Proceedings of the 2nd Workshop on European Scientific and Industrial Collaboration, Newport, 1 – 3 September 1999. 1999. UK: Mechatronics Research Centre, University of Wales College, Newport. pp. 207 - 214.
- [7] Fujimura K., 1993, *Motion planning in time-varying domains: the case of cyclic motions*, Advanced Robotics, Vol. 7, No. 5, pp491-505.
- [8] Reif J. and Sharir M., 1985, *Motion Planning in the Presence of Moving Obstacles*, IEEE 26th Annual Symposium on Foundations of Computer Science, pp. 144-154.
- [9] Fujimura K., 1991, *Motion Planning in Dynamic Environments*, Springer – Verlag.
- [10] Hart P. E., Nilsson N. J. and Raphael B., *A Formal Basis for the Heuristic Determination of Minimum Cost paths*, IEEE Transaction on Systems Science and Cybernetics, July 1968, Vol. 4(2), p. 100-107.
- [11] Alexopoulos C. and Griffin P. M., March 1992, *Path Planning for Mobile Robot*, IEEE Transaction on Systems, Man and Cybernetics, Vol. 22, pp. 318-322.

Mr Anastasios Diamantopoulos, Professor Geoff N. Roberts and Dr David J. Harwood
Mechatronics Research Centre
University of Wales College, Newport
Allt-yr-yn Campus. PO Box 180, Newport, NP19 5XR. United Kingdom
TEL: +44 1633 432487, FAX: +44 1633 432442. e-mail: a.diamantopoulos@newport.ac.uk

A HEURISTIC ALGORITHM FOR MOTION PLANNING OF AN AGV IN DYNAMIC ENVIRONMENTS

Abstract: *In this paper an algorithm for planning safe motion for an autonomous guided vehicle (AGV), in a two-dimensional environment populated by time-depended obstacles is presented. It is assumed that the time depended obstacles are moving along linear paths with constant velocity. All information about the obstacles, such as, shapes, locations and velocities are given ahead planning. The AGV is a point-robot with bounded velocity modulus. A heuristic algorithm is proposed for finding time-minimal motion for the AGV in such dynamic environments, in $O(n^2 \log n)$ computational time, where n is the total number of the obstacles' vertices.*

1 Introduction

Over the years, a number of methods have been developed for solving the problem of planning collision-free paths for an AGV, in a two-dimensional environment, populated by stationary obstacles, from an initial point to a goal point. One of the first methods is the VGRAPH approach, which was developed by Lozano-Pérez and Wesley, 1979. This approach uses the concept of Visibility Graph to solve the aforementioned problem, in $O(n^2 \log n)$, computational time, where n is the total number of the C-obstacles' edges. The Visibility Graph is an important combinatorial structure, El Gindy and Avis, 1981, Guibas et al, 1986, Ghosh and Mount, 1987, which has been extensively used for the Robot Motion Planning problem.

In this paper, an algorithm for finding a time-minimal motion in a two-dimensional environment, populated by time varying obstacles, from an initial point to a goal point (this problem is also referred in the literature as *two-dimensional asteroid avoidance problem*) using the concept of visibility graph is presented.

2 Previous work survey

Sutner and Maass, 1988 have solved the problem of motion planning in a dynamic environment populated by one-dimensional time varying obstacles. In their approach they introduced another dimension into the configuration space, so the one-dimensional moving obstacles were described as polygonal objects in a space-time. The motion planning of a point-robot with bounded speed in one dimension was solved in polynomial time on the total number of vertices of the polygonal space-time obstacles.

In Canny and Reif, 1987, it has been shown that motion planning for a point-robot, with a bounded velocity modulus, in a two-dimensional environment populated by arbitrarily many

moving, non-rotating convex obstacles, that move at constant velocity is NP-hard. Erdmann and Lozano-Pérez, 1987 consider the problem of planning motion for multiple robots. They assign priorities to each robot and then they plan the motion of one robot at a time according to its priority. The configuration space-time was constructed for each moving object in order to represent the constraints imposed on it by its time-depended environment. A motion was then obtained by using vertex-to-vertex translation of the point-robot in configuration space-time. Kant and Zucker, 1986 consider monotonous path in space-time by decomposing the trajectory planning problem into two subproblems: (i) planning a path to avoid collisions with static obstacles and (ii) planning the velocity along the path to avoid collisions with moving obstacles. Fujimura and Samet, 1993 presented an algorithm to find a motion for a point-robot, in an environment populated by time-depended obstacles and destination point. The environment's obstacles were convex polygons, which move in a fixed direction at constant speed. The algorithm they proposed finds the time-minimal motion given that the point-robot moves faster than the obstacles and the destination point, in polynomial time using the accessibility concept. Fujimura, 1993 proposed two algorithms for solving motion planning problem for a point robot that moves in an environment where the moving obstacles and the destination point have cyclic motion. These algorithms are the *hit-and-leave*, which is suited for sensor-based navigation and the *accessibility* algorithm, which is more suited when the environment is accurately known ahead planning. Both of these algorithms establish a collision-free motion, providing that the robot moves faster than the obstacles. The motion defined by the second method is also time-minimal. Fujimura, 1994, proposed an algorithm for motion planning of a point-robot in an environment with transient obstacles and destination point. This algorithm was based on the propagation of a wave-front technique and finds the time minimal motion between two points in the plane in polynomial computational time.

Reif and Sharir, 1985, showed that the problem of motion planning in a three-dimensional environment populated by moving obstacles is a PSPACE-hard problem, when the robot's velocity modulus is bounded and NP-hard when the robot's velocity modulus is not bounded.

The approach presented in this paper finds the time-minimal vertex-to-vertex motion for a point robot between two points, in an environment populated by moving obstacles, in polynomial time. This approach is based on the V*MECHA algorithm previously reported by the authors, Diamantopoulos A. et al., 1999.

3 Formulation of the problem

The formulation of the problem is as follows. Let $W=R^2$ be the workspace, populated by two-dimensional polygonal obstacles P_i , where $i \in N_+$, the AGV's start point S and its goal point G . The AGV A is a point-robot, which translates freely with bounded velocity modulus. Therefore the AGV's configuration space $C = W = R^2$ and every obstacle's configuration space $CP_i = P_i$. Some of the obstacles of W are allowed to translate along linear paths at fixed orientation, with constant velocity v_{P_i} , between two time instances t_1 and t_2 . Let $P_i(t)$ denote the region of W occupied by P_i at instance t , ($t \geq 0$) and $CP_i(t)$ denote the region of C occupied by CP_i at instance t . Before the start and after the end of the motion of every moving obstacle P_i , the obstacle has velocity equal to zero. The interiors of any two obstacles are not allowed to overlap at any time. The problem is to determine a collision-free time-minimal motion for the AGV A , from the start point S to the goal point G , given that the AGV A is constantly moving with its maximum velocity ($\max(v_A)$) and that the description

of the obstacles (such as shapes, locations and velocities) are accurately known ahead of planning.

4 Configuration Space-Time

The motion planning problem, is a more difficult problem than the path planning problem, where its solution is just the establishment of a sequence of (collision-free) points connecting the start point to the goal point for a robot or the determination that such sequence does not exist. In dynamic environments the solution of the motion problem is not just about spatial reasoning, because the parameter of time should be taken into consideration. Since there are time-dependent obstacles in the environment, a continuous function of time should be defined in order to specify the AGV's position in the environment at all times. This can be achieved by adding another dimension (the time dimension) to the AGV's configuration space C . The new configuration space with the extra dimension is called configuration space-time, CT and is defined as, $CT = C \times [0, +\infty)$. Every CP_i from C (either static or moving) maps into configuration space-time CT as a static obstacle CTP_i . Since the interior of the CP_i s do not overlap in C at all times, then the interior of the CTP_i s do not overlap in CT at all times. In our problem the configuration space $C = \mathbb{R}^2$, so the $CT = C \times [0, +\infty)$ and hence $CT = \mathbb{R}^2 \times [0, +\infty)$. The two-dimensional CP_i s from C map to CT in prisms. Figures 1a and 1b illustrate the relation between two-dimensional configuration space C and the three-dimensional configuration space-time CT .

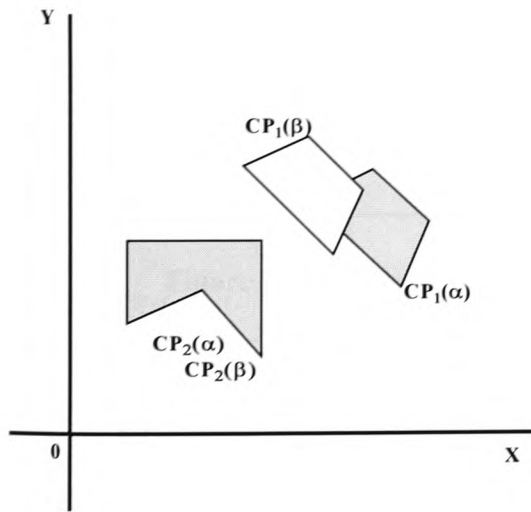


Figure 1a This figure illustrates the positions of CP_1 and CP_2 in two time instances α , β in C .

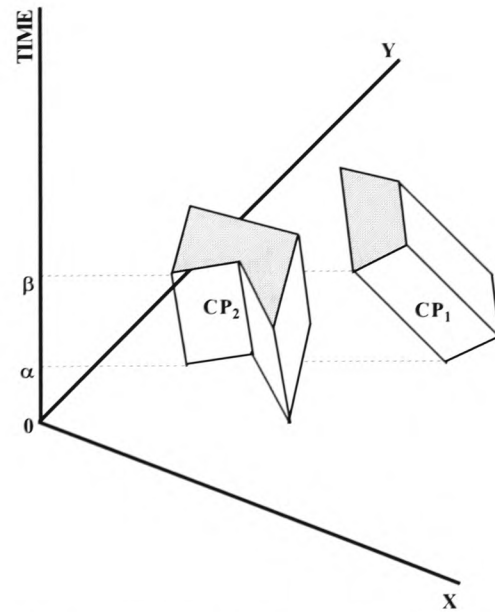


Figure 1b This figure illustrates the way the CP_1 and CP_2 map from C into CT in static prisms.

As it can be noticed in Figures 1a and 1b, the static obstacles in C correspond to orthogonal to x - y plane prisms in CT and the time varying obstacles from C correspond to sloped with respect the x - y plane prisms in CT . The slope of each obstacle in CT , is determined by its constant linear velocity. Note, that all the edges of the CTP_i s in CT which do not constitute the bases of the prisms, correspond to vertices of CP_i s in C , these edges are called *shaft*

edges. The edges of the CTP_is that constitute the prisms' bases in CT correspond to the edges of the CP_is in C. The start point S and the goal point G of C correspond to half-lines in CT, which emanate from S and G respectively and are parallel to the time-axis. The configuration space-time CT (which is static) can be searched for a collision-free path between the S and G. Note that since no object can move back in time, the path in CT should be strictly monotone in time. In the next section some possible extensions on the V*MECHA algorithm, are presented to enable it to find time-minimal motion for an AGV in dynamic environments.

5 Extension of V*MECHA for time-varying domains

According to the assumptions made earlier, the AGV A (point-robot) moves with constant velocity $\max(v_A)$. Therefore, the set of all the reachable configurations, from a specific configuration p in C, between two time instances t_α and t_β (with $t_\alpha < t_\beta$), is defined by the perimeter of a circle CR, with centre p and radius $r = v_A (t_\beta - t_\alpha)$, Figure 2.

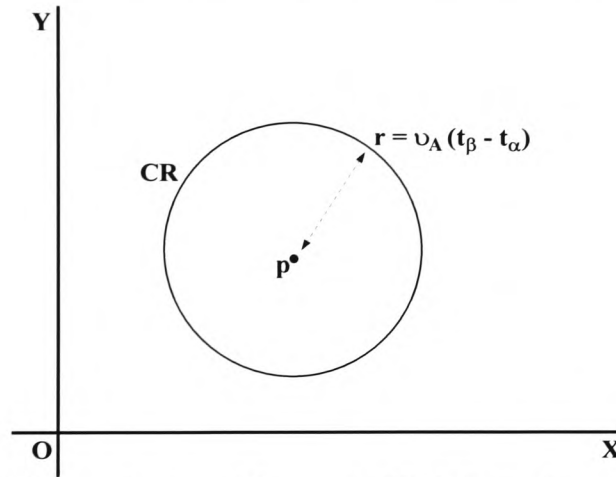


Figure 2 The perimeter of the cycle CR defines the set of all reachable configurations from the configuration p in C for an AGV which is moving with constant velocity v_A , between time t_α and t_β .

In the three-dimensional CT, the set of all reachable configurations for the AGV A from a specific configuration p , between two time instances t_α and t_β is defined by the surface of a right cone CN emanating from configuration p , given that A is moving with constant velocity. The configuration p is the apex of the cone. The height h of the cone is equal to $t_\beta - t_\alpha$. The radius R of the base of the cone is equal to $v_A (t_\beta - t_\alpha)$. The angle ϑ created by the slant height ℓ of the cone and the plane Q (Q is defined by the apex of the cone and is parallel to the x-y plane of CT) is defined as follows:

$$\tan \vartheta = \frac{h}{R} \Rightarrow \tan \vartheta = \frac{t_\beta - t_\alpha}{v(t_\beta - t_\alpha)} \Rightarrow \tan \vartheta = \frac{1}{v} \Rightarrow \tan \vartheta = v^{-1} \Rightarrow \vartheta = \tan^{-1}(v^{-1})$$

Figure 3 illustrates the cone whose surface defines the set of all the reachable configurations from a configuration p in CT.

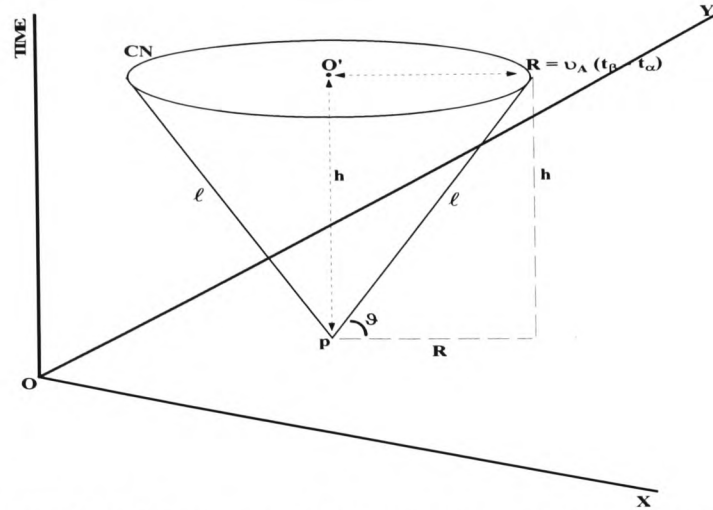


Figure 3 The surface of the cone CN defines the set of all reachable configurations from configuration p in the three-dimensional space-time for an AGV A, which is moving with constant velocity v_A , between time t_α and t_β .

Any intersection between the surface of the cone CN and any shaft edge or the goal half-line in CT is a candidate configuration for the AGV to move to. The AGV moves to such configuration depending on whether this configuration is visible from the apex of the cone or not and whether it is on the path traced by the time-minimal motion. An intersection corresponds to a vertex of an obstacle CP_i or to the goal point G in C at some time t.

Summarizing, if a ray v is swept about p by keeping constant angle $\theta = \tan^{-1}(v^{-1})$ with the plane Q, all the p-visible points (points that can get connected with p without any prior interceptions) at angle θ in the CT can be identified. These points correspond to vertices of the CP_i s in C at some time t and more specific to p-visible vertices in C, which are used to construct a kind of a visibility graph. Alexopoulos and Griffin, 1992, used a cone, in order to identify all the visible vertices from given vertex. Then they proposed an algorithm for finding a greedy in time motion for a robot in dynamic environments. The algorithm was called E*GRAPH and it was based on the V*GRAPH algorithm. The E*GRAPH algorithm seems to be incorrect, because it is based on the V*GRAPH which has been proven to be incorrect Conn et al., 1997. Fujimura, 1994 also used the idea of the cone in order to identify the collision fronts of transient obstacles in an environment, and then he proposed an algorithm for finding a time-minimal motion for an AGV in environments populated with transient obstacles and destination point.

According to two properties in Diamantopoulos et al., 1999, the shortest Euclidean path in a two dimensional static environment, can not contain non-extreme vertices of a *p-visible sequence* (set of consecutive p-visible vertices on a single CP_i) and *obtuse polygon vertices*, are also applicable in dynamic environments. Since the AGV is moving with constant velocity it can be easily shown that within the time-minimal motion the AGV does not pass through vertices rejected by the above properties. The two properties are applied in the projection of the CTPs to the plane Q. These properties are used in order to reduce the number of vertices which will be examined for the construction of the time-minimal motion.

D*MECHA is a heuristic search algorithm guided by the evaluation of the function \hat{t}_f over every vertex of the visibility graph and finds the time-minimal motion from the start node S

to the goal node G . The function $\hat{t}_f(n)$ at any node n is an estimation of $t_f(n)$ which defines the time-cost of travelling along the path traced by the actual time-minimal motion from S to G constrained to pass through node n . More formally $\hat{t}_f(n) = \hat{t}_g(n) + \hat{t}_h(n)$, where $\hat{t}_g(n)$ is the time-cost of travelling along the path traced by the time-minimal motion from S to n the algorithm found so far, estimating $t_g(n)$, which defines the time-cost of travelling along the path traced by the actual time-minimal motion from S to n . The function $\hat{t}_h(n)$ is an estimation of the function $t_h(n)$ which defines the time-cost of travelling along the path traced by the actual time-minimal motion from node n to a preferred goal of n (in this case G). However an estimation of the function $t_h(n)$ is not easy to find, so the best way to define it is to rely on information about the problem domain. In time-minimal applications when the AGV travels with constant speed, a good estimation of $t_h(n)$ is $\hat{t}_h(n)$, the time of travelling along the airline distance from n to the goal node. This distance is the smallest possible distance between these nodes, so $\hat{t}_h(n)$ is lower bound of $t_h(n)$. When $\hat{h}(n)$ is lower bound the D*MECHA is admissible. The algorithm D*MECHA finds the time-minimal vertex-to-vertex motion for an AGV in a dynamic environment and is stated as follows:

D*MECHA algorithm

begin

$\theta := \arctan(v^{-1})$

put S **in** Motion;

put S **in** Open;

mark S visited;

$\hat{t}_g(S) := 0$;

while (Open \neq nil) **do**

begin

$w := \{i \in \text{Open} : \hat{t}_f(i) < \hat{t}_f(j) \mid \forall j \in \text{Open}, \text{ resolve ties arbitrarily but always in favor of the goal node}\};$

remove w **from** Open;

if $w = G$ **then exit while loop**;

$VV := \{w\text{-visible points at angle } \theta\};$

$EV := \{\text{extreme vertices of the } w\text{-visible sequences in the projections of } VV \text{ into } Q\};$

mark all the non-extremes vertices useless

$AV := EV - \{\text{obtuse vertices in } EV\};$

for each vertex $i \in AV$ **do**

if i **is not marked** useless **then**

if i **is not marked** visited **then**

begin

$\hat{t}_h(i) := t(d_{\text{air}}(i, G))$ (the time of travelling on the airline distance from i to G);

$\hat{t}_g(i) := \hat{t}_g(w) + t(d(w, i));$

$\hat{t}_f(i) := \hat{t}_g(i) + \hat{t}_h(i);$

put i **in** Motion **with pointer toward** w ;

put i **in** Open;

mark i visited;

end;

else if $\hat{t}_g(i) > \hat{t}_g(w) + t(d(w, i))$ **then**

begin

```

    redirect pointer of i toward w in Motion;
    if i ∈ Open then remove i from Open
     $\hat{t}_g(i) := \hat{t}_g(w) + t(d(w, i));$ 
     $\hat{t}_f(i) := \hat{t}_g(i) + \hat{t}_h(i);$ 
    put i in Open;
    end;
end;
if (Open ≠ nil) then return Motion by tracing all the pointers backward from G to S
else return failure;
end.

```

In the algorithm, *Motion* is a spanning tree, which represents at any instant the best motion obtained so far. For each visited node n (except S) a pointer to its parent is held. The function $\hat{t}_f(n)$ is associated with each node n in the current *Motion*. The $t(d(w, n))$ is a function, which represents the time-cost of travelling along an edge, which connects two vertices in the visibility graph, with $\max(v_A)$. The $d_{air}(w, n)$ is a function, which represents the time-cost of travelling, with $\max(v_A)$ along the airline distance between nodes w and n . The list *Open* contains at any instant, the vertices that are candidates for expansion next. All the vertices of the environment are initially marked as unvisited and useful.

6 Analysis of the algorithm

The D*MECHA algorithm, finds the time-minimal vertex-to-vertex motion for an AGV (point-robot), in a two-dimensional dynamic environment, from a point S to a point G , in $O(n^2 \log n)$ computational time, where n is the total number of C-obstacles' vertices. Indeed, by analyzing the algorithm it can be noticed that the steps that find the n with smallest \hat{t}_f in *Open* and remove it from the *Open* requires $O(n)$ time, since there are, at most n vertices in *Open*, the treatment of its children requires also $O(n)$ time. The step for computing the set visible vertices for a κ vertex at angle θ with the x-y plane, requires $O(n \log n)$ time. The computation time for finding the extreme vertices in a κ -visible sequence is $O(n)$. The obtuse visible vertices can be removed in time $O(n)$. Since all these steps appear nested in the while loop of the algorithm, which can be repeated at most n times, the overall time complexity of D*MECHA is $O(n^2 \log n)$. The proof of D*MECHA's admissibility and optimality is similar to V*MECHA's and is presented in Diamantopoulos A. et al., 1999.

7 Discussion

Under the assumption that the AGV is moving with constant velocity equal to $\max(v_A)$, the motion that is established by the D*MECHA algorithm is a time-minimal motion. During this motion the AGV is moving from S to G through CP_i 's vertices in a non-stop manner. If only stationary obstacles populate the environment then the time-minimal motion established by D*MECHA defines also the shortest path from S to G . In the case that the AGV is allowed to move in a non vertex-to-vertex manner without constant velocity, then the D*MECHA algorithm does not always return the time-minimal motion but a greedy in time motion. This happens because it is possible for the time-minimal motion, to be defined in such a way that the AGV will have to stop at a specific place to wait for a moving obstacle to move out of its way and then start moving again towards the goal position. In this kind of

motion it is possible for the AGV to reach the point G without having to pass through C-obstacles' vertices.

8 Acknowledgements

The authors would like to thank Antonio Zirilli for the discussions during the preparation of this manuscript and for his useful comments.

9 References

- Alexopoulos C. and Griffin P. M., March 1992, *Path Planning for Mobile Robot*, IEEE Transaction on Systems, Man and Cybernetics, Vol. 22, pp. 318-322.
- Canny J. and Reif J., 1987, New Lower Bound Techniques for Robot Motion Planning Problems, IEEE 28th Annual Symp on Foundations of Computer Science, pp. 49-60.
- Conn R. A., Elenes J. and Kam M., August 1997, *A Counterexample to the Alexopoulos-Griffin path planning Algorithm*, IEEE Transactions on Systems, Man and Cybernetics, Vol. 27, No. 4, pp. 721-723.
- Diamantopoulos A., Roberts G. N. and Harwood D., 1999, *A new hybrid approach for path planning of an AGV*, 2nd International Symposium Advanced Manufacturing Processes, Systems and Technologies, Bradford, UK, pp 199-208.
- El Gindy H. and Avis D., 1981, *A Linear Algorithm for Computing the Visibility Polygon from a Point*, Journal of Algorithms, Vol. 2, pp. 186-197.
- Erdmann M. and Lozano-Pérez T., 1987, *On multiple moving objects*, Algorithmica, Vol. 2, No. 4, pp. 477-522.
- Fujimura K., 1993, *Motion planning in time-varying domains: the case of cyclic motions*, Advanced Robotics, Vol. 7, No. 5, pp491-505.
- Fujimura K., 1994, *Motion Planning Amid Transient Obstacles*, The International Journal of Robotics Research, Vol. 13. No. 5, pp 395-407.
- Fujimura K. and Samet H., 1993, *Planning a Time-Minimal Motion Among Moving Obstacles*, Algorithmica, Vol. 10, pp 41-63.
- Ghosh S. K. and Mount D. M., 1987, *An Output Sensitive Algorithm for Computing Visibility Graphs*, IEEE 28th Annual Symp. on Foundations of Computer Science, pp. 11-19.
- Guibas L., Hershberger J., Leven D., Sharir M. and Tarjan R. E., 1986, *Linear Time Algorithms for Visibility and Shortest Path Problems Inside Simple Polygons*, ACM Symposium on Computational Geometry, pp 1-13
- Kant K. and Zucker S. W., 1986, *Toward Efficient Trajectory Planning: The Path-Velocity Decomposition*, The Internat. Journal of Robotics Research, Vol. 5, No.3, pp 72-89.
- Lozano-Pérez T. and Wesley M. A., 1979, *An Algorithm for planning collision-free paths among polyhedral obstacles*, Communicat. of the ACM, Vol. 22, No. 10, pp. 560-570.
- Reif J. and Sharir M., 1985, *Motion Planning in the Presence of Moving Obstacles*, IEEE 26th Annual Symposium on Foundations of Computer Science, pp. 144-154.
- Sutner M. and Maass W., 1988, *Motion planning among time dependent obstacles*, Acta Informatica, Vol. 26, pp. 93-122.

A NEW HYBRID APPROACH FOR PATH PLANNING OF AN AGV

Anastasios Diamantopoulos, Geoff N. Roberts and
David J. Harwood

Mechatronics Research Centre
University of Wales College, Newport
Allt-yr-yn Campus
PO Box 180
Newport
NP9 5XR, UK

Tel: int +44 (0)1633 432487, Fax: int +44 (0)1633 432442
e-mail: a.diamantopoulos@newport.ac.uk

Synopsis

In this paper, an algorithm for planning a safe path for a two-dimensional polygonal Autonomous Guided Vehicle (AGV), which translates freely without rotation in a two-dimensional known environment, populated by physical stationary obstacles, is presented. The obstacles are transformed to represent the locus of forbidden positions of an arbitrarily chosen reference point on the AGV and the AGV is considered as a point-robot (reference point). The V*MECHA algorithm is then applied to find the shortest safe path between an initial and goal point for the AGV in the environment in $O(n^2 \log n)$ computational time, where n is the total number of the transformed obstacles' vertices.

Notations

$O(n)$	denotes the computational complexity of an algorithm.
Γ	is an operator, which when applied to a node n_i in a graph takes the value $\{(n_j, c_{ij})\}$, which is a set of pairs for all n_i 's successors, where c_{ij} is the cost of the edge that connects node n_i to its successor node n_j .
\overline{kx}	denotes the line segment kx .
$d(x, y)$	is the Euclidean distance between node x and y connected by an edge.
$d_{air}(x, y)$	is the airline Euclidean distance between node x and y .

1 INTRODUCTION

The use of AGVs or articulated robot arms, for moving objects, tools or devices from one point to another and performing tasks in the workspace of a manufacturing environment, is an important consideration for the efficiency of an automated manufacturing process. The problem of finding the shortest collision-free path for an AGV between two points in an environment populated by static obstacles has been solved correctly, by Lozano Pérez and Wesley (1). In their approach they compute the obstacles' configuration space (C-Obstacles) and then they construct the visibility graph by connecting all the mutual visible vertices of the C-Obstacles and the AGV's initial and goal points. When the visibility graph has been constructed it is then searched for a path from the initial to the goal point. This algorithm is

called VGRAPH and finds the shortest safe path in $O(n^2 \log n)$ computation time, where n is the total number of the C-Obstacles' edges.

Alexopoulos and Griffin in (2) proposed an algorithm named V*GRAPH for solving the planar stationary-obstacle problem for polygonal obstacles and AGV, in $O(n^2 \log n)$, where n is the total number of C-Obstacles' vertices. The V*GRAPH algorithm was meant to be quicker than VGRAPH for an average case even though they have the same time complexity. This improvement is achieved due to the fact that V*GRAPH does not construct the entire visibility graph like VGRAPH, but only a part of it. However Conn et al. (3) show with a counterexample that V*GRAPH approach is not global and therefore incorrect.

In this paper a hybrid approach similar to V*GRAPH is presented. This approach employs both V*GRAPH and A* (shortest path algorithm using heuristics Hart et al. (4)) algorithms and solves the planar stationary-obstacles path planning problem in worst case $O(n^2 \log n)$ computation time, where n is the total number of C-Obstacles' vertices. The advantage of this approach over the other algorithms is that much smaller number of vertices are selectively considered for the creation of the visibility graph and fewer vertices are examined for the construction of the path. Since the visibility graph is constructed by using less edges the algorithm is quicker for the average case and therefore improves manufacturing efficiency.

2 A BRIEF DESCRIPTION OF V*GRAPH ALGORITHM

The basic idea behind the V*GRAPH algorithm is that the number of vertices that are considered for the construction of the visibility graph is smaller. This implies that fewer nodes are examined for the identification of the shortest path and therefore the procedure is quicker. The reduction of the number of nodes that are considered for the construction of the visibility graph is achieved in V*GRAPH by only expanding a subset of visible nodes that would have been expanded by VGRAPH. This reduction is due to the fact that not all the vertices are suitable enough to be candidates for expansion in order to construct the visibility graph. In the V*GRAPH algorithm the vertices with interior polygon angle greater than π radians and the non-extreme vertices of a visible sequence are not considered for the construction of the visibility graph. All the vertices in an environment that are visible from a vertex κ are called κ -visible vertices. The set of consecutive κ -visible vertices on every C-Obstacle _{i} is called κ -visible sequence.

The V*GRAPH algorithm uses these two conditions to minimize the size of the visibility graph and then using a heuristic strategy guided by the evaluation of a function \hat{f} finds the shortest path between s (initial point) and g (goal point). The value of $\hat{f}(n)$ over every vertex n of the visibility graph estimates the shortest Euclidean distance from s to g through node n . Specifically this approach borrows the $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$, function from the A* search algorithm and uses it in the same way as the A* algorithm (where $\hat{g}(n)$ estimates the actual cost of the shortest path from s to n and $\hat{h}(n)$ estimates the actual cost of the shortest path from n to g). Alexopoulos and Griffin (2) presented their algorithm as follows:

V*GRAPH algorithm

Begin

P :={s};
 Open :={s};
 $\hat{g}(s) : = 0$;


```

 $\hat{f}(s) := 0;$ 
repeat
   $w := \{i \in \text{Open} : \hat{f}(i) \leq \hat{f}(j) \mid \forall j \in \text{Open}\}$ 
  remove  $w$  from Open;
  put  $w$  in  $P$ ;
   $VV := \{w\text{-visible vertices not in Open}\};$ 
   $EV := \{\text{extreme vertices of the } w\text{-visible paths in } VV\};$ 
   $AV := EV - \{\text{obtuse vertices in } EV\};$ 
  for each vertex  $i$  in  $AV$  do
     $\hat{h}(i) := \text{Euclidean distance } d(i, g) \text{ from } i \text{ to } g;$ 
     $\hat{g}(i) := \hat{g}(w) + d(w, i);$ 
     $\hat{f}(i) := \hat{g}(i) + \hat{h}(i);$ 
    put  $i$  in Open;
  until  $((\text{Open} = 0) \text{ or } (w = g));$ 
  if  $(\text{Open} = 0)$  then
    exit with failure;
  if  $(w = g)$  then
    exit with  $P$ ;
end.

```

The overall time complexity of this algorithm is $O(n^2 \log n)$, where n is the total number of C-Obstacles' vertices.

3 A COUNTER-EXAMPLE ON V*GRAPH ALGORITHM

Conn et al. (3) in 1996 used a counter-example on V*GRAPH algorithm in order to prove its incorrectness. The scenario of the counter-example was as follows; consider the following planar stationary-obstacle environment and a point-robot, which needs to move from s (initial point) to g (goal point) by following the shortest (Euclidean) path.

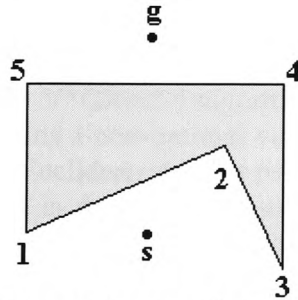


Fig 1 The planar stationary-obstacle environment used in the counter-example.

The coordinates of the obstacle's vertices, initial point and goal point were given as follows:

$$\begin{aligned}
 s &= (3, 1), & g &= (3, 4), & 1 &= (0, 1), & 2 &= (4, 2) \\
 3 &= (5.5, 0), & 4 &= (5.5, 3), & 5 &= (0, 3)
 \end{aligned}$$

Since the robot is a point-robot, the C-Obstacle is the same as the obstacle of the environment and the algorithm can be applied to find the shortest path between s and g . When the V*GRAPH algorithm is applied to the above scenario then the following table with values of the algorithm's variables is obtained while the algorithm is tested.

Table 1 This table summarizes the values of the algorithm's variables while it is tested.

Iterations	i	j	P	Open	w	VV	EV	AV	h(i)	g(i)	f(i)
			s	s							
1			s	\emptyset	s	1,2,3	1,3	1,3			
	1			1					4.24	3	7.24
	3			1,3					4.72	2.69	7.41
			s,1	3	1	2,5	2,5	5			
2	5			3,5					3.16	5	8.16
3			s,1,3	5	3	1,2,4	1,2,4	1,4			
	1			1,5					4.24	8.28	12.52
	4			1,4,5					2.69	5.69	8.38
4			s,1,3,5	1,4	5						

What can be noticed from Table 1, is that at the forth iteration of the 'repeat' loop of the algorithm, the path $P = \{s, 1, 3, 5\}$ intersects the boundary of the obstacle (Figure 2).

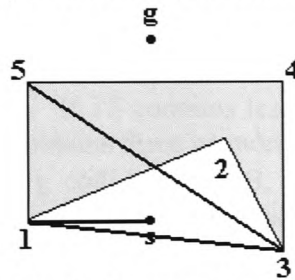


Fig 2 At this figure the intersection of the path (bold line) with the obstacle's boundary is illustrated.

Since a valid solution for the above scenario exists, which is $P = \{s, 1, 5, g\}$ and the algorithm does not return this, but returns an invalid one instead, then the algorithm is not global and therefore incorrect. The V*GRAPH algorithm is also not optimal, because the solution of the above scenario contains a non-optimal vertices' subsequence $\{\dots, 1, 3, 5, \dots\}$, which should not be included in the Euclidean shortest path.

The incorrectness of V*GRAPH is due to the misuse of A*. In the Alexopoulos and Griffin (2) paper it is stated that V*GRAPH employs A* algorithm for finding the shortest path, however this claim is not correct. The problem of V*GRAPH is the fact that all the vertices it expands are contained in the final path. This happens because the algorithm is not capable of removing any vertex from the path, so each vertex is placed in the path it is not possible to be removed by the algorithm even if a vertex of a better path should be placed instead. A very important step of A* algorithm is missing from V*GRAPH, this step is the one which allows the algorithm to keep a record of pointers for the vertices of the path and to update it at each iteration according to the best path found so far. The facts presented above lead to the result that V*GRAPH is not complete, not optimal and it does not employ A* which is complete and optimal.

4 THE V*MECHA ALGORITHM

To overcome the deficiencies of V*GRAPH a new hybrid approach (V*MECHA) for path planning of an AGV in a two-dimensional stationary-obstacle environment is proposed. The V*MECHA algorithm is a completion of V*GRAPH algorithm. This algorithm is a mixture of the V*GRAPH and A* algorithms. V*MECHA minimize the size of visibility graph using Theorems 1 and 2 presented below and finds the shortest path between s (initial point) and g (goal point) using powerful heuristics from A*. V*MECHA starts from the initial node s and generates a part of the visibility graph by applying recursively the successor operator Γ . A node is said to be expanded when the operator Γ has been applied to it. The value of the operator Γ when applied to a node n_i is a set of pairs $\{(n_j, c_{ij})\}$, where c_{ij} is the cost of the edge which connects node i to node j in the graph, for all the successor nodes j 's of the node i . During the execution of the algorithm, each time a node is expanded a successor w of the node, the cost of the optimal path for reaching w and a pointer to the predecessor of w are stored. When the algorithm reaches the node g , then it is terminated and no more nodes are expanded. The shortest path is then obtained by tracing backwards all the pointers from g to s . If an algorithm is guaranteed to return a shortest path between two nodes in a graph that all edges have greater than zero length, then it is called admissible.

Theorem 1: From a vertex κ , only the extreme vertices on a κ -visible sequence need to be considered for the construction of the visibility graph because any line segment from κ to g through the extremes is the shortest.

Proof: Suppose VS is a κ -visible sequence of a CB (CB is the configuration space of an obstacle B) in the environment. If VS contains less than three vertices then these are the extremes. Suppose now that VS contains three or more vertices and the extremes are x and y . If a line segment connecting κ to g collide with CB, then a shortest path passing through k will contain either the line segment $\overline{\kappa x}$ or the line segment $\overline{\kappa y}$. Otherwise will contain a polygonal line with endpoints one of the two extremes and therefore it is not the shortest. The extremes of a κ -visible sequence can be computed in $O(n)$ computational time.

Theorem 2: A shortest collision-free path from s to g cannot contain vertices with interior polygon angle greater than π radians (obtuse vertices).

Proof: If the shortest path, $Path$ from s to g contains an obtuse vertex q of the CB, then if the predecessor and the successor of q on $Path$ is respectively x and y , then it exists a point v on \overline{qy} excluding q and y which is visible from x (Figure 3).

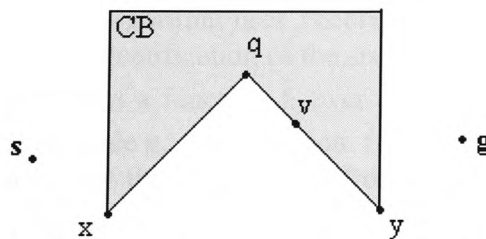


Fig 3 Illustration of Theorem 2, where the path containing an obtuse vertex is $\{s, x, q, y, g\}$ and the actual shortest path is $\{s, x, y, g\}$.

Using the triangle inequality,

$$d(x, v) < d(x, q) + d(q, v)$$

it is obtained that,

$$d(x, v) + d(v, y) < d(x, q) + d(q, v) + d(v, y)$$

but since,

$$d(x, q) + d(q, v) + d(v, y) = d(x, q) + d(q, y)$$

then,

$$d(x, v) + d(v, y) < d(x, q) + d(q, y)$$

The above inequality implies that if the path passes through v instead of q is shorter, which contradicts the fact that *Path* is the shortest path. This proof by contradiction leads to the conclusion that the shortest path should not include obtuse vertices.

To demonstrate the effectiveness of Theorem 1 and 2 consider the planar environment of Figure 4, where the C-Obstacle is denoted by CB and s, g are the initial and goal points respectively.

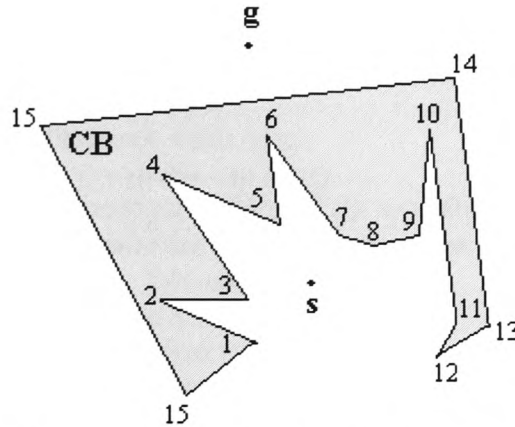


Fig 4 This figure illustrates the great reduction of vertices considered for the construction of the visibility graph from $\{1, 3, 4, 5, 6, 7, 8, 9, 11, 12\}$ to $\{1, 3, 9, 12\}$.

Applying Theorem 1 to the above example, the s -visible vertices to be included in the visibility graph have been reduced to 1, 3, 9, 11, 12. If Theorem 2 is now used over the set of vertices obtained as a result of Theorem 1, the vertices included in the visibility graph are 1, 3, 9, 12. This example illustrates the great reduction of vertices considered for the construction of the visibility graph from $\{1, 3, 4, 5, 6, 7, 8, 9, 11, 12\}$ to $\{1, 3, 9, 12\}$.

The V*MECHA algorithm defines a function \hat{f} for every node in the graph in such a way, that it determines which node will be expanded next. This is an efficient way to ensure that nodes that are not on the optimal path are not expanded, and nodes that should be on the optimal path are not ignored. The algorithm uses Theorems 1 and 2, to minimize the number of vertices to be examined for the identification of the shortest safe path, and using a heuristic strategy guided by the evaluation of a function \hat{f} over every vertex, finds the shortest path from the start node s to the goal node g . The function $\hat{f}(n)$ at any node n is an estimation of $f(n)$ which defines the actual cost of the shortest path from s to g constrained to pass through node n . More formally $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$, where $\hat{g}(n)$ is the cost of the shortest path from s to n , the algorithm found so far estimating $g(n)$, which defines the actual cost of the shortest path from s to n . The function $\hat{h}(n)$ is an estimation of the function $h(n)$ which defines the actual cost of the shortest path from node n to a preferred goal of n (in this case g). However

an estimation of the function $h(n)$ is not easy to find, so the best way to define it is to rely on information about the problem domain. In shortest path applications a good estimation of $h(n)$ is $\hat{h}(n)$, the airline distance from n to the goal node. This distance is the smallest possible distance between these nodes, so $\hat{h}(n)$ is lower bound of $h(n)$. When $\hat{h}(n)$ is lower bound the V*MECHA is admissible. The V*MECHA algorithm stated as follows:

V*MECHA algorithm

begin

put s **in** Path;

put s **in** Open;

mark s visited;

$\hat{g}(s) := 0$;

while (Open \neq nil) **do**

begin

$w := \{i \in \text{Open} : \hat{f}(i) < \hat{f}(j) \mid \forall j \in \text{Open}, \text{ resolve ties arbitrarily but always in favor of the goal node}\}$;

remove w **from** Open;

if $w = g$ **then exit while loop**;

$VV := \{\text{w-visible vertices}\}$;

$EV := \{\text{extreme vertices of the w-visible paths in } VV\}$;

mark all the non-extremes vertices useless

$AV := EV - \{\text{obtuse vertices in } EV\}$;

for each vertex $i \in AV$ **do**

if i **is not marked** useless **then**

if i **is not marked** visited **then**

begin

$\hat{h}(i) := \text{Airline distance } d_{\text{air}}(i, g) \text{ from } i \text{ to } g$;

$\hat{g}(i) := \hat{g}(w) + d(w, i)$;

$\hat{f}(i) := \hat{g}(i) + \hat{h}(i)$;

put i **in** Path **with pointer toward** w ;

put i **in** Open;

mark i visited;

end;

else if $\hat{g}(i) > \hat{g}(w) + d(w, i)$ **then**

begin

redirect pointer of i **toward** w **in** Path;

if $i \in \text{Open}$ **then remove** i **from** Open

$\hat{g}(i) := \hat{g}(w) + d(w, i)$;

$\hat{f}(i) := \hat{g}(i) + \hat{h}(i)$;

put i **in** Open;

end;

end;

if (Open \neq nil) **then return** Path **by tracing all the pointers backward from** g **to** s

else return failure;

end.

The above algorithm finds the Euclidean shortest path for a two-dimensional AGV in a two-dimensional stationary-obstacle environment from a start point s to a goal point g . In the algorithm *Path* is a spanning tree, which represents at any instant the best path obtained so far, for each visited node n (except s) a pointer to its parent is held. The function $\hat{f}(n)$ is associated with each node n in the current *Path*, this is an estimation function of $f(n)$ which is the actual cost of the best path from s to g constrained to pass through n . The $d(w, n)$ is a function, which represents the cost of an edge, which connects two vertices in the visibility graph. The $d_{air}(w, n)$ is a function, which represents the cost of the airline distance between node w and n . The list *Open* contains at any instant, the vertices that are candidates for expansion next. All the vertices of the environment are initially marked as unvisited and useful.

4.1 The admissibility of V*MECHA

The evaluation function is $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$, where $\hat{g}(n)$ is the minimum cost path from s to n , found by V*MECHA so far, and $\hat{h}(n)$ is an estimation of the optimal path from n to a preferred node of n in a graph which all edges have greater than zero length.

Theorem 3: If $\hat{h}(n) \leq h(n)$ for all n , then V*MECHA is admissible, Hart et al., (4).

Since in V*MECHA, $\hat{h}(n)$ is the airline distance between node n and g , then is always less than or equal to $h(n)$, therefore the V*MECHA is admissible.

4.2 The optimality of V*MECHA

According to the knowledge of the problem domain, the precision of the heuristic function can be tuned. For instance if $\hat{h}(n) = 0$, this means that no heuristic information is available. However the algorithm will still find the minimum cost path if there is one, because $\hat{h}(n) \leq h(n)$. An algorithm $Algo_1$ is more informed than another algorithm $Algo_2$, if the heuristic information which underestimating $h(n)$ used by $Algo_1$ is greater than the one used by $Algo_2$. A less informed algorithm will expand more nodes than a more informed algorithm. With a restriction on \hat{h} , it is possible to show that V*MECHA is optimal in a sense that it does not expand more nodes than any other admissible algorithm, which is less than or equally informed as V*MECHA. The restriction used on \hat{h} is:

$$\hat{h}(n_i) - \hat{h}(n_j) \leq d(n_i, n_j)$$

The above inequality restricts the difference of the estimated path costs from any two nodes to the goal node, to be less than or equal, to the actual cost of the shortest path between the nodes if there is one. If the heuristic information is applied consistently to every node this assumption is true. For this reason it is called consistency assumption. For the proof of V*MECHA's optimality, in a sense that it does not expand more nodes than any other less or equally informed, admissible algorithm the following two theorems are necessary to used.

Theorem 4: If the consistency assumption is satisfied and a node n is expanded by V*MECHA, then $\hat{g}(n) = g(n)$.

This theorem was used by Hart et al., (4) for the proof of A*'s optimality. It shows that when the heuristic information of the problem is applied consistently at all nodes and the node

n has been expanded by V*MECHA algorithm, then the cost of the estimated shortest path from node s to n found by the algorithm, is guaranteed to be equal to the actual cost of the shortest path from node s to n .

Theorem 5: If \hat{h} is an underestimation of h , then for every node n expanded by V*MECHA, $\hat{f}(n) \leq f(s)$.

Proof: If n is a node which has been expanded from V*MECHA and n is the goal then $\hat{f}(n) = f(s)$ and the theorem is proved. If n is not the goal, then it is known from (4) that just before n was expanded by V*MECHA there was a node n' on the optimal path with $\hat{f}(n') \leq f(s)$. If $n = n'$ then the theorem is proved otherwise n would have been expanded by V*MECHA instead of n' and the following inequalities should be true,

$$\hat{f}(n) \leq \hat{f}(n') \leq f(s)$$

therefore the theorem is proved. This theorem shows that if the heuristic information \hat{h} about the problem domain underestimates h , then the estimated cost of the path from s to g through n for any n expanded by V*MECHA is less than or equal to the actual cost of the shortest path from s to g . Now the optimality of V*GRAPH can be proved.

Theorem 6: If V*MECHA is more than or equally informed as another admissible algorithm *BMECHA* and the consistency assumption is satisfied, then if a node n is expanded by V*MECHA, it is also expanded by *BMECHA*.

This theorem shows that V*MECHA is optimal in a sense that no other admissible algorithm which is less than or equally informed as V*MECHA, expands less nodes than V*MECHA. This theorem was also proved and used by Hart et al., (4) for the proof of A*'s optimality.

4.3 The time complexity of V*MECHA

The V*MECHA algorithm, finds the shortest path for a two-dimensional robot, in a two-dimensional stationary-obstacle environment, from a point s to a point g in $O(n^2 \log n)$ computational time, where n is the total number of C-Obstacles' vertices. Indeed, by analyzing the algorithm it can be noticed that the steps that find the n with smallest $\hat{f}(n)$ in *Open* and remove it from it require $O(n)$ time, since there are at most n vertices stored in *Open*. The step for computing the set visible vertices for a κ vertex requires $O(n \log n)$ time, de Berg M. et al. (8). The computation time for finding the extreme vertices in a κ -visible sequence is $O(n)$, Alexopoulos and Griffin (2). The obtuse visible vertices can be removed in time $O(n)$. The treatment of κ 's children requires $O(n)$. Since all these steps appear nested in the while loop of the algorithm, which can be repeated at most n times, the overall time complexity of V*MECHA is $O(n^2 \log n)$. The V*MECHA algorithm appears to have the same time complexity as VGRAPH. However Wesley (5), Asano et al. (6) and Edelsbrunner (7), proposed algorithms which construct the visibility graph in $O(n^2)$ and then by using Dijkstra's shortest path algorithm $O(n^2)$, obtain the shortest path from s to g in overall time complexity of $O(n^2)$.

The advantage of V*MECHA over the other algorithms that construct the entire visibility graph, is that in an average case the size of the visibility graph is greatly reduced and the algorithm can be faster. An example of such a case is illustrated in Figure 5.

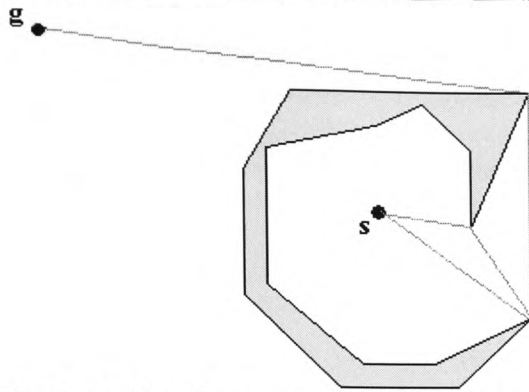


Fig 5a This figure shows the reduced number of edges construct the visibility graph by V*MECHA.

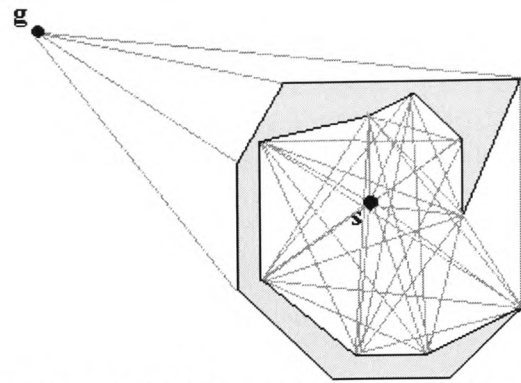


Fig 5b This figure shows the number of edges construct the visibility graph by the other algorithms.

The above case is an average case in which it is clear that V*MECHA algorithm considers a much smaller number of vertices for the construction of the visibility graph in comparison to any other algorithm which constructs the entire visibility graph and therefore is quicker.

5 CONCLUSIONS

In this paper a new algorithm has been presented for mobile robot path planning. The algorithm is called V*MECHA and finds the shortest-distance, collision-free path for a mobile robot in an environment populated by stationary obstacles. V*MECHA was developed to overcome the deficiencies of a previous algorithm, the V*GRAPH. The time complexity of V*MECHA is $O(n^2 \log n)$ and its advantage over other similar algorithms is that in an average case it is quicker because it expands a smaller number of vertices for the construction of the visibility graph.

References

- (1) Lozano-Pérez T. and Wesley M. A., An Algorithm for planning collision-free paths among polyhedral obstacles, Communic. of the ACM, 1979, Vol. 22, No. 10, p. 560-570.
- (2) Alexopoulos C. and Griffin P. M., Path Planning for Mobile Robot, IEEE Transaction on Systems, Man and Cybernetics, March 1992, Vol. 22, p. 318-322.
- (3) Conn R. A., Elenes J. and Kam M., A Counterexample to the Alexopoulos-Griffin path planning Algorithm, IEEE Transactions on Systems, Man and Cybernetics, August 1997, Vol. 27, No. 4, p. 721-723.
- (4) Hart P. E., Nilsson N. J. and Raphael B., A Formal Basis for the Heuristic Determination of Minimum Cost paths, IEEE Transaction on Systems Science and Cybernetics, July 1968, Vol. 4(2), p. 100-107.
- (5) Welzl E., Constructing the Visibility Graph for N Line Segments in $O(n^2)$ Time, Information Proceeding Letters, 1985, Vol. 20, p. 167-171.
- (6) Asano Tak., Asano Tes., Guibas L., Hershberger J. and Hiroshi I., Visibility-Polygon Search and Euclidean Shortest Paths, 26th IEEE Symposium of Foundations on Computer Science, Portland, OR, 1985, p.155-164.
- (7) Edelsbrunner H., Algorithms in Combinatorial Geometry, 1987, Monographs on Theoretical Computer Science, Springer-Verlag, Berlin, p. 275-278.
- (8) de Berg M., van Kreveld M., Overmars M., Schwarzkopf, Computational Geometry algorithms and applications, 1997, Springer.