1

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

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

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

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

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

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

AVIS

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

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

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

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

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

**An Integrated World Modelling and Path Planning
System for a Three-Dimensional Workspace**

by

Andrew Moizer, B.Eng.

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Masters of Engineering

Department of Systems and Computer Engineering,
Carleton University,
Ottawa, Ontario, Canada

April 20, 1987

© Andrew D. Moizer 1987

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

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

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

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

ISBN 0-315-46288-4

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

*An Integrated World Modelling and Path Planning
System for a Three Dimensional Workspace*

submitted by Andrew David Moizer, B.Eng., in partial fulfilment of the requirements for the degree of Masters of Engineering.

Chairman,
Department of Systems and
Computer Engineering

Thesis Supervisor

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
May 25, 1987

Abstract

This thesis addresses the requirements and design of an onboard global navigation system suitable for an autonomous underwater inspection vehicle. A integrated planning and world modelling system for three dimensional workspaces has been designed and implemented for a vehicle performing inspection/cleaning tasks within an offshore steel jacket platform. Free space is modelled by closely packed convex polyhedral cells joined together by specific 'doorways' in the faces of adjacent cells. Individual path segments are given weighting factors that indicate the paths relative merit. The path planner rapidly finds optimal paths between arbitrary points within the workspace. The planner uses a powerful branch & bound search algorithm that incorporates a variable 'search factor' to control the focus of the search and improve performance. A prototype system has been implemented in Prolog and tested on a Vax 11/750 and a Motorola MC68000 based personal computer.

Acknowledgements

My long stay at Carleton has been enjoyable and enlightening but also long enough to tax my memory when attempting to remember the many people who helped make this thesis possible.

The first group of people that must be thanked are all my fellow graduate student colleagues that have come and gone during my stay. It is these people who contributed so much to the fertile environment in which we worked. It is a shame that there is insufficient space to mention them all. Thanks people!

I would also like to thank the many professors that I had dealings with during my stay. I must especially thank my long suffering supervisor Dr. Bernie Pagurek who offered guidance, wisdom, encouragement and funding throughout, without losing his composure or faith in me. Words cannot express my gratitude.

Staying within the University I must thank the office and support staff within the Systems Engineering Department that kept the system rolling and 'suffered' so long with me. My desk is finally available! I would also like to acknowledge the Graduate Studies Department personnel whom I now know so well.

There were many people from outside the university that contributed to this thesis. These include the personnel at the Systems Lab at NRC, the employees of Robotics Systems International in Victoria for their insight into the undersea world, the Brown & Root and Hudson Engineering companies for their offshore platform engineering drawings and NSERC for their funding that is so important.

I must also thank my parents and family for the support and encouragement that they have given me throughout my scholastic career that has enabled me to get to where I now am.

Finally, words cannot express the feelings I have towards a very special person who arrived after the start of this scholastic escapade and never wavered in her emotional (and financial) support, my ever suffering wife Susan. It is to her, and our son Christopher who represents the future, that I would like to dedicate this thesis. Thanks, I hope I have done you both proud.

Andrew Moizer, April 11, 1987.

Contents

1	Introduction	1
1.1	The Underwater Environment	2
1.2	A Tetherless Submersible	4
1.3	An Overview of the Navigation System	6
1.4	Thesis Outline	8
2	Description of the Problem	10
2.1	Offshore Structures and Inspection	10
2.2	The Underwater Vehicle	13
2.2.1	Physical Construction and Layout	13
2.2.2	The Onboard Computer System	13
2.3	Subsystem Description	16
2.3.1	Mission Module	16
2.3.2	Global Navigation	17
2.3.3	Local Navigation	18
2.3.4	Vehicle Control	18
2.3.5	Vehicle Position	19
2.3.6	Toolset Control	19
2.4	The Global Navigation Problem	21
3	World Modelling	23
3.1	Requirements of the Model	23
3.2	Modelling Techniques	24
3.3	The Three-dimensional Modelling Technique	26
3.3.1	A Modelling Example in Two Dimensions	28
3.3.2	A Modelling Example in Three Dimensions	29
3.4	More Modelling Details	35
3.4.1	The Start and Goal Points	35
3.4.2	Link Scaling	37
3.5	Automatic Generation of the Model	37
3.5.1	Platform Definition	38

3.5.2	Model Generation	39
3.6	Updating the Model and Machine Learning	39
3.7	Discussion	40
4	Path Planning	42
4.1	Requirements of the Planner	42
4.2	Planning Techniques	42
4.3	Path Planner Design	44
4.3.1	The $g(n)$ Function	44
4.3.2	The $h(n)$ Function	45
4.4	The Nature of the Path	47
4.4.1	Optimization	47
4.5	Planning Issues	48
4.6	Discussion	49
5	Prototype System	50
5.1	Path Planner Implementation	51
5.2	Performance	51
5.2.1	Planning Time	60
5.3	Personal Computer Performance	61
5.4	Prototype System Code Overview	62
5.4.1	Implementation Approaches	62
5.4.2	The World Model	63
5.4.3	The Path Planner	64
5.4.4	Other Modules	66
5.4.5	Code Size	66
5.5	Integration into an Underwater Vehicle	66
6	Conclusions	68
6.1	Results	68
6.1.1	Modelling	69
6.1.2	Planning	70
6.2	Extensions and Future Work	70
6.3	Contributions	72
6.4	Conclusion	72
	References	73
	A Prolog Source Code Listings	79
	B Platform Drawings	103

List of Figures

1.1	A Typical Offshore Steel Jacket Platform	3
1.2	A Typical ROV fitted with a Manipulator	5
1.3	Block Diagram of the Control System of an Autonomous Underwater Vehicle	7
2.1	North Sea Jacket Platform	12
2.2	Experimental Tetherless Submersible	14
2.3	Inspection Vehicle Attached to Platform Structure	20
3.1	Two Space Model Development	30
3.2	Network Facts for Two Space Model	31
3.3	Three Space Modelling Example	33
3.4	Three Space Trajectory Example	34
5.1	The Large Test Platform Structure	53
5.2	Program Run, Search Factor of 1.2	54
5.3	Program Run, A* Search	55
5.4	Small Test Structure, showing Simple Path	56
5.5	Program Run for Figure 5.4	57
5.6	Small Test Structure, showing Complex Path	58
5.7	Program Run for Figure 5.6	59
5.8	Model Generation Output for Large Test Platform	60

Chapter 1

Introduction

The development of autonomous mobile robots, capable of operation within a wide ranging environment, is an area of intense and varied research. It has been argued that the development of autonomous mobile vehicles will lead to machines that exhibit a form of general intelligence [36], closer to that exhibited by warm blooded animals. True general purpose machines of this type have yet to be developed. Experience obtained in a more controlled environment is a good way to reach this level of competence. There are many mobile robots under development that operate in more constrained environments performing specific tasks. A few are outlined below, roughly in order of increasing generality and difficulty.

- 'floorwalking' mobile robots, operating in a two-dimensional environment of rooms and obstacles [35,22,12].
- undersea vehicles designed for operation within a platform structure, performing inspection type tasks [26,46,6,25].
- longer range autonomous underwater vehicles for underice mapping or pipeline inspection [48].
- autonomous land vehicles, typically for military applications [39].
- airborne vehicles, includes aircraft, helicopters and missiles [17].

The development of systems such as those outlined above will lead both to more advanced autonomous mobile robots and to more intelligent machines in general.

1.1 The Underwater Environment

The underwater environment is a harsh world that man is poorly equipped to cope with. He has been designing machines to operate underwater for many centuries now. The last three decades have seen an increased interest in smaller, unmanned vehicles designed to perform specific underwater tasks. With the increasing interest (and reliance) on undersea oil and mineral reserves man has developed many aids that allow him to function underwater. However as it is still a harsh and dangerous work area there is a great attraction in machines that can perform tasks remotely, allowing man to stay above the surface. This thesis deals with part of such a machine.

The offshore oil industry requires the construction and maintenance of extremely large structures to support drilling and production equipment. These include large steel jacket platform structures consisting of large tubular piles supported by extensive framing and bracing members. These structures are permanently anchored to the ocean floor with piles driven deep into the bottom. Figure 1.1 shows the principal components of a typical eight leg offshore platform.

These structures must be inspected periodically to ensure safety and to meet government and industry regulations. At the present time these inspections are conducted by divers. Saturation diving techniques must be used in order to operate to the depths common with many newer platforms. Diving is expensive and dangerous and the cost and risks increase dramatically as the operating depth is increased. Platforms are now being constructed in depths over 1000 feet [47], which is approaching the limit for divers, and deeper installations are planned [28].

Figure 1.1: Principal Components of a Typical Offshore Steel Jacket Platform
(from [18])

There is an obvious need for a machine that could replace divers, reducing both the costs and the risks [10].

A wide variety of unmanned undersea vehicles have now been built [6] and are being used more often for support to offshore operations [11]. The majority of these submersibles are classed as *remotely operated vehicles* (ROV's) since they are controlled from the surface by a human operator. An umbilical cable (or tether) supplies power and passes control signals between the ROV and the support vessel. The ROV's are equipped with video cameras and powerful lights to enable the operator to see where he is and what he's working on. Larger ROV's are often equipped with one or more manipulators which allow more complex tasks to be performed. A typical ROV is shown in figure 1.2.

The usefulness of an ROV is often limited by the tether which tends to become entangled in the work area and has a high drag that limits the range, speed and maneuverability of the vehicle. Entanglement is a severe problem when working near a jacket platform and usually prohibits an ROV from operating within the structure at all. Obviously this poses a severe problem if a complete remote inspection of a jacket structure is to be conducted. A solution to this problem is to remove the tether and use an autonomous submersible [50]. It is likely that most new vehicles designed for remote underwater inspection will operate autonomously since the tether represents too great a limitation [19]. However this introduces a host of new problems and requirements, which will be discussed in the next section.

1.2 A Tetherless Submersible

When the tether is removed from an ROV many changes must be made to the vehicle. The vehicle must now carry sufficient energy to run the propulsion system, tools and computers for the entire mission. The loss of the tether also cuts off

Figure 1.2: A Typical ROV fitted with a Manipulator (from [45])

7

almost all the communication between the vehicle and the surface, as the acoustic channel has low bandwidth and reliability [15]. Substantial onboard intelligence must be added to provide the mission planning and execution information previously supplied by the human operator. Meeting the intelligence requirements of the vehicle is a challenging task that presses the bounds of present portable, real-time hardware and software.

The onboard computer system can be logically and physically separated into modules dedicated to performing specific vehicle functions. This approach is flexible and extensible and is sympathetic to the varying real-time requirements of the different subsystems. A typical system consists of a command (mission) module that controls other modules for navigation, vehicle control, positioning, tool control etc. This structure is illustrated in figure 1.3.

Of these modules the mission control and global navigation portions most directly replace the human operator of an ROV. The mission module coordinates the actions of all the vehicle systems towards the completion of a mission goal. The global navigation system provides the high level, goal directed planning functions previously provided by the human operator. The planner must be as general as possible and capable of solving a wide variety of route selection problems if a flexible autonomous vehicle is to be produced. The global navigation portion of the vehicle control system is addressed in this thesis.

1.3 An Overview of the Navigation System

The global navigation system of an autonomous submersible is responsible for the mission or global, goal oriented path planning. It requires a comprehensive world model which accurately describes the operating environment. The model must be concise, complete and include all the features of the environment and their attributes. The system can then use the information in the model to plan optimal

Figure 1.3: Block Diagram of the Control System of an Autonomous Underwater Vehicle

safe trajectories between arbitrary locations specified by the mission computer.

A three-dimension planning and modelling system has been developed for vehicle performing inspection/cleaning tasks within an offshore steel jacket platform. This system represents a solution to a *real world* problem in autonomous vehicles and machine intelligence. The system was not designed specifically for any existing (or planned) underwater vehicle but rather was developed in a manner that would allow easy adaptation to a specific installation. The underlying assumptions on which the system was based are all realistic and justifiable for the assumed target environment. Throughout the development emphasis was placed on obtaining a realistic and workable solution to the problem of autonomous vehicle navigation.

The navigation system developed was implemented in the Prolog programming language [9,8], specifically the MProlog version developed by Logicware [29] which allows modular program development. The language allows rapid prototyping and lends itself to the fact based, network modelling technique and to the heuristic search procedure used to find paths through the structure.

1.4 Thesis Outline

In the following chapter more information on the offshore operating environment and structures, offshore operations and a more detailed description of the problem will be presented. Sufficient details on ROV's and the performance required to complete the necessary tasks will be given to understand the system requirements. The assumptions and constraints that were applied to the design will be outlined and justified.

Chapter 3 addresses the area of world modelling as it pertains to this problem. The requirements of a system for an autonomous underwater structure inspection vehicle will be outlined. Previous applicable work will be summarized. A full description of the modelling technique developed, its features and limitations will

be presented. Aspects of machine learning applicable to this problem will be outlined.

Chapter 4 considers the path planning problem and the requirements of this system. The approach developed will be presented and demonstrated.

Chapter 5 deals with the implementation of the modelling and path planning techniques presented earlier. Examples of the system in operation, modelling different environments and performance figures will be presented. Dynamic aspects and system integration into a physical vehicle will be considered.

Chapter 6 concludes and summarizes the material presented. Extensibility and suitability to other problems are outlined. Finally the direction of future research and areas where further work might be appropriate are alluded to.

Following this are a series of appendices that contain the majority of the Prolog source code and engineering drawings of an offshore platform designed and built for use in the Gulf of Mexico.

Chapter 2

Description of the Problem

This chapter provides more concrete details about the area of investigation. Details of the offshore environment and the associated equipment, structures and requirements will be given to provide the necessary background for the research. It will lead to a more precise statement of the problem that was studied. The assumptions and constraints that were applied to the design will be outlined and justified.

2.1 Offshore Structures and Inspection

An offshore steel jacket platform is a large, heavy complex structure. Figure 1.1 showed the principal components of a typical eight leg drilling platform designed for use in the Gulf of Mexico. The framework must be strong enough to withstand normal and storm environmental conditions and must provide sufficient room for drilling and production operations. The main vertical leg members also provide a template through which piles are driven deep into the sea bed to support the platform.

The overall dimensions of the platform designed for a water depth of 300 feet would be about 140 by 40 feet at the surface and 200 by 110 feet at the seafloor. The main piles are typically four feet in diameter and are driven 300 feet into

the seafloor. The leg members are several inches larger with wall thicknesses of around one inch. The bracing members range from one to three feet in diameter with wall thicknesses of $1/4$ to $3/4$ of an inch. Overall the jacket will weigh in the order of 2000 tons. Detailed drawings of such a platform, provided courtesy of the Brown & Root International Inc. [5], are contained in appendix B. Another platform, designed for use in the North Sea, is shown in figure 2.1 to give an idea of the complexity of these structures.

The newer platforms being built for deeper water are much larger and heavier. The *Cognac* [47] platform, designed for a depth of 1000 feet is 400 by 380 feet at the seafloor and 165 by 85 feet at deck level. The platform weighs 33,500 tons with 24 seven foot diameter piles driven 450 feet into the seafloor. This platform will support the drilling of 62 wells using two drilling rigs simultaneously.

As mentioned earlier these structures must be inspected regularly to ensure safety and to meet government and industry regulations. This inspection is generally divided into three categories as follows [16,32]:

1. a general visual survey of the structure (usually by TV) to discover obvious damage and to verify structural integrity.
2. a local inspection of some of the key structural nodes to check for hidden defects. This requires cleaning the node (welds) of marine growth followed by a detailed visual and photographic examination.
3. additional non-destructive testing (NDT) (eg. ultrasonic) of welds following the above inspection to check for cracks too small to be detected by the unaided eye.

Each of the above three inspections will require a different vehicle task description and toolset, which will be discussed in a later section. All the tasks share the requirement of a vehicle that can operate within the platform and navigate

Figure 2.1: North Sea Jacket Platform(from [16])

from place to place with consistency and accuracy. Thus the basic navigation requirements of the vehicle will remain similar regardless of the inspection task being carried out.

2.2 The Underwater Vehicle

This section discusses the actual vehicle in more detail. The vehicle components and their physical layout will be outlined as will the structure and functionality of the onboard computer system.

2.2.1 Physical Construction and Layout

The physical layout of a typical autonomous underwater inspection vehicle is similar to the ROV that was shown in figure 1.2. The vehicle consists of an open framework that the remaining components are attached to. There are a number of propeller thrusters to provide propulsion and attitude control and a power supply unit. A sealed, pressure resistant vessel is attached to contain and protect the computers and related electronics. An appropriate toolset and manipulator for the mission being performed would also be attached. Finally syntactic foam buoyancy is added to the top of the frame to give the vehicle a slight positive buoyancy and a stable righting moment. The size of a typical vehicle would range from six to ten feet long and four to six feet wide and high. The vehicle would weigh between 500 and 4000 pounds in air, depending on size and capabilities. The layout of a smaller research autonomous vehicle is shown in figure 2.2.

2.2.2 The Onboard Computer System

The onboard computer system is responsible for all vehicle control and mission planning functions. It must replace the skills of the human operator required to

pilot an ROV. This will require advanced onboard computing power to perform the many tasks. There are several subsystems that can be identified, many of which can be designed to operate more or less independently and in parallel. These include:

- mission planning, control and monitoring.
- global *high level* navigation.
- local navigation and collision avoidance.
- vehicle control: speed, depth, heading, etc.
- vehicle position.
- toolset control.

This separation along functional boundaries also lends itself to a system that is implemented by multiple computers operating in parallel, each responsible for a separate subsystem. The basic layout of the control system was shown in figure 1.3. Modern high speed micro-processors can be used to implement this approach, providing powerful computers with low power, size and cost. This approach of splitting the onboard computer system into functional modules and utilizing multiple micro-processors has been used by many unmanned autonomous vehicles developed to date [24,23,48,51,14,22,44]. Using this approach the computer system can be split along the functional divisions outlined in figure 1.3 with a processor dedicated to each major function.

The development of systems that will fulfill these requirements can proceed in parallel in many cases. This requires that the interaction between these systems be well defined, or can at least be realistically assumed. Given that this requirement can be met the module design can be attempted.

2.3 Subsystem Description

In order to define the interactions between the modules that were outlined above, it is necessary to describe what each one does in more detail. This will be done in the next sections.

2.3.1 Mission Module

This module represents the 'highest level' in the computer system hierarchy. It most closely replaces the functionality that was supplied by the human operator of an ROV. The module coordinates the actions of the other computer system modules, controls the execution of the mission, monitors the progress and deals with unexpected events that occur during the mission. The module contains the intelligence and judgement required to keep the vehicle operating under changing operating conditions, vehicle failures etc. Expert system and knowledge engineering techniques have been investigated for incorporation into this level of the control system [2]. New methods such as these appear necessary in order to replace the judgement and information formally supplied by the human operator.

The module contains a complete description of the vehicles mission in functional terms rather than a specific plan or path to be followed. This approach is very flexible since it allows the specific path of the vehicle to be changed during the mission as required. For an inspection task the mission description would include a list of the areas to be examined, the nature of the inspection to be performed and details on how the inspection is to be conducted. Specific details of the inspection area could also be included, as a supplement to the world model, or could be left to the lower level modules.

The mission computer could also receive commands from the surface through the use of acoustic signals sent through the water. However this channel has a very low bandwidth and cannot be relied upon, especially within a platform

structure where there tend to be numerous echos and reflections [15]. This does give a method through which simple high level commands can be issued during operation, such as abort mission or proceed to a specific station. This approach has been used to provide limited control to several underwater vehicles under development [32,48].

The mission module operates in an asynchronous manner in order to deal with the widely varying response times of the other modules and to retain maximum flexibility.

2.3.2 Global Navigation

The global navigation module is responsible for planning and defining paths for the vehicle to follow from one point in the workspace to another. In general these paths are relatively complex and the nature of the environment must be considered in order to produce a satisfactory solution. Thus the navigation module must contain a *global world model* and a *path planner* to generate a suitable path that the vehicle can follow. The world model contains an accurate, concise and complete description of the operating environment in a form that can be used by the planner. The planner utilizes the model and additional information in conjunction with an efficient searching procedure to plan an optimal, safe path for the vehicle to follow.

The navigation module receives commands from the mission computer that specify the start and goal points of the path required. The path is returned to the mission computer in a form that can be used by the local navigation module, usually in the form of *waypoints* that the vehicle should pass through.

The navigation module would operate with a cycle time in the range of 30 seconds to several minutes when planning any reasonably long path. Providing that the mission computer is planning ahead to some extent this is a sufficient 'real time' response.

2.3.3 Local Navigation

The local navigation module is responsible for guiding the vehicle between relatively closely spaced points sent from the mission computer. This module plans a path between these points and then controls the vehicle to keep it on this path, within some reasonable tolerance. This module would monitor the vehicle's progress through information received from the vehicle position module and the vehicle control module. The output would consist of commands that are issued to the vehicle control module to move the vehicle along the path. Essentially this subsystem operates as a closed loop controller with limited planning capabilities.

The local navigation module often contains a simple collision avoidance system that can quickly deal with unexpected obstacles if they are detected. The path replanning at this level is on a very local level that would not affect the mission plan. The replanning at this level can be considered akin to the reflex systems in animals.

This module would operate in a fairly tight cycle of planning followed by commands sent to the vehicle control system. The collision avoidance would operate in an 'interrupt' style that would initiate local replanning whenever an appropriate sensory input was received.

2.3.4 Vehicle Control

The vehicle control system is responsible for the actual 'low level' control of the vehicle. It receives commands for specific speeds and headings and converts them into control signals for the various vehicle thrusters. The module contains the information of the vehicle's dynamics necessary to accurately control the vehicle. The whole system operates as a typical real time controller with cycle times in the 0.1 second range.

2.3.5 Vehicle Position

This module would be responsible for determining the actual coordinate position of the vehicle. For a vehicle operating within an offshore platform structure this is usually done with a sonar system with several fixed transponders located outside the platform. Measuring time delays from the different sensors allows a position to be determined. The depth of the vehicle can be determined more accurately by a water pressure reading. This system is complicated by the many reflections and echos that can occur from the various members of the platform. However if a sufficient number of transponders are used a reliable position can be determined [21,37].

2.3.6 Toolset Control

The module controls the tools that actually do the inspection and cleaning operations. For a visual inspection, only verification of the correct node and camera control would be involved. For a cleaning and detailed photographic inspection task, more sophisticated control would be required and in most cases the toolset could be considered a separate robot system. The most promising approach for a detailed inspection is a toolset incorporating manipulators that can anchor the vehicle by grasping the platform in the inspection area. A diagram showing this approach is shown in figure 2.3.

In most cases the toolset control system operates independently of the other onboard computer systems. Specialized manipulator systems are under development that are designed to perform an automated inspection of a platform structure with little or no human intervention [32,53].

Figure 2.3: Inspection Vehicle Attached to Platform Structure (from [32])

2.4 The Global Navigation Problem

Now that the vehicle systems have been described and their interactions defined a more formal description of the global navigation system can be given. This definition represents the design specification for the experimental modelling and planning systems developed for this thesis.

The requirements of the global navigation system are as follows:

1. the input to the system consists of two arbitrary positions within the structure, specified in three-dimension coordinates, representing the starting and goal points of the trajectory to be planned.
2. the output from the system consists of a path from the start to the goal, described by a list of *waypoints* that the vehicle should pass through.
3. the system should plan an optimal (in terms of distance, hazard or complexity) path that avoids the platform structure, preferably with a wide safety margin.
4. a comprehensive world model that accurately describes the operating environment should be maintained. The model should be simple and structured in a form that could incorporate additional information if required.
5. the system should plan the path in the minimum possible time, typically less than 1 minute.
6. the design should be rugged, flexible and extensible.
7. interactions with other modules of the onboard computer system should be minimized.
8. the navigation system is not responsible for the actual execution of the path (although it should be planned in a manner to simplify its implementation).

This is left to other portions of the onboard computer system.

9. an accurate position of the vehicle is assumed to be available at all times, allowing the vehicle to use a path specified only by waypoints in three-dimension coordinates.

Chapter 3

World Modelling

This chapter discusses the world modelling aspect of the navigation system for an autonomous vehicle. The technique developed to meet the three-dimensional requirements of the underwater environment will be presented and demonstrated. Finally, enhancements and automatic generation will be discussed.

3.1 Requirements of the Model

The choice of the modelling technique is critical to the design and implementation of a successful navigation system. The fundamental operating requirements of the modelling system were outlined in section 2.4. There are two aspects to the modelling problem, the conceptual representation of the environment and the machine implementation of the model. The specific details of how the modelling system and the planner are implemented will be deferred until chapter 5. The key to a successful modelling technique lies in defining a conceptual model that adequately describes the operating environment *for the tasks being performed* and lends itself to a computer implementation. In general the final 'internal' representation of the model should be a network or graph of nodes and links. This form readily lends itself to a wide variety of searching techniques, a feature that greatly simplifies the path planning stage of navigation.

3.2 Modelling Techniques

Numerous techniques have been used to model two and three-dimensional environments in systems designed for mobile robots and fixed manipulators. In general these techniques can be divided into two categories: those that represent the obstacles and those that represent free space directly. The common technique when the obstacles are modelled is to use the *configuration space* approach [31,35]. In this case the object to be moved (or the vehicle) is shrunk to a reference point and all the obstacles are expanded by an amount according to the shape of the object. Free space is then taken as paths between the vertices of the *grown* polygonal obstacles. This technique has been used successfully in two dimensions if the object has not been allowed to rotate. When three dimensions are considered the procedure of growing the obstacles becomes much more complex, and if rotations are allowed then a different configuration space is required for each rotation angle. A disadvantage of this approach is that the paths that are planned tend to skirt the obstacles closely rather than staying in open free space.

There are several approaches that can be taken if the free, unoccupied, space is modelled directly. Most of the effort centres around the choice of which geometric shapes are best suited to model free space for the particular application in question. Brooks [3,4] uses generalized cones consisting of a medial axis and left and right radii as defined by a sweeping rule. Paths are found along these axes, referred to as freeways, with transitions between cones taking place at their intersections. A further development of this concept is the use of convex polygons to model the intersection region of the cones and freeways [27]. This allows more freedom in the transition between freeways. An advantage of this method in general, is that the paths follow the centre of the free space giving the obstacles a wide berth. This technique is more difficult to apply to a three space problem as the cone axes are no longer guaranteed to intersect and the cone generation and sweeping rules

C

become much more complex.

Another approach, as taken by Ruff and Ahuja [43], developed for three-dimension modelling involves dividing the workspace into small cubes or voxels which are labelled full if the voxel is part of an obstacle. Free space is then modelled by inverting the voxel description of the obstacles which is then converted to a graph representation through a medial axis transformation. This graph can then be searched to find a path through the obstacles. This approach works well but computational costs rise significantly as the number of voxels is increased, as is the case for large models.

Free space can be modelled directly by convex polygons, an approach that has successfully been used in two-dimensional spaces by both Crowley [12,13] and Chatila [7]. In this approach free space is divided into closely packed convex polygonal cells, chosen since a path between any two points in the cell must lie totally within it and thus must remain in free space. The cells are linked together by specific doorways or corridors that join the faces of adjacent polygons that are large enough to accommodate the vehicle. A path through the workspace can then be found by moving from one cell to another through the appropriate doorways.

None of the above techniques are well suited for modelling an underwater platform structure. The three-dimensional nature of the problem precludes the direct use of methods developed for 2-D work and the extra degrees of freedom inherent in three space make a direct extension difficult. The large size of the structure compromises the use of techniques that divide the workspace into small blocks as the high number of cells cause problems in computation speed and storage. Configuration space methods become increasingly difficult to compute and store as the number of degrees of freedom increases from two, as is the case in this problem. These factors indicate that a different modelling technique is required. The method developed, which incorporates some of the concepts presented above applied to a three space model, will be presented in the following sections.

3.3 The Three-dimensional Modelling Technique

The offshore platform structure is large and relatively uncluttered with obstacles. The members generally follow a regular grid pattern and the framework is triangulated in the main planes of the structure. The vehicle will naturally pass through these triangles when moving within the structure. It makes sense to exploit these features when considering the modelling technique design.

The technique that was developed and implemented divides the platform structure solely into (three-dimensional) *convex polyhedra* of free space. These polyhedral cells have the property that a line segment joining any two points within the cell remains totally within the cell. A vehicle can then follow a straight path between any safe entry and exit points of a cell (providing they are on different faces), a feature that greatly simplifies path planning. This approach is well suited to modelling a large offshore platform as free space can generally be divided into large cells, a feature that keeps the size, complexity and cost of the model reasonable. An offshore oil platform can be adequately described using a fairly small set of polyhedra types, for example: tetrahedra, pyramids, prisms and cubes. Having divided the structure into cells an interconnection mechanism must be designed and implemented.

If closely packed polyhedral cells are used to represent the free space, in general, each face of one cell is also a portion of a face of the neighbouring cell. If the intersection is large enough for the vehicle to pass through freely, then a *doorway* linking the two cells can be added to the model. When all the possible doorways between the cells have been established, the connectivity of the free space has been determined and represented.

In the method that was implemented a doorway was located at the centroid of the common face between the two adjacent cells. The centroid of the face was chosen as it is a simple point that is removed from the bracing members that

define the common face. In an offshore platform these common faces are usually triangular, due to the bracing patterns, so a doorway is located at the centroid of all bracing triangles common to two cells. In general the doorways should be chosen as far away from obstacles as possible.

This representation can be further developed to reduce the amount of storage and to produce a form of the model particularly suited for path planning. Since a straight line path through the cell is determined by its endpoints, a network structure can be built up consisting only of the doorways and the links between them. This network can be developed directly from the information of the cells and the interconnecting doorways. The model can now be stored as a list of the positions of all the doorways and a list of the links that join together two doorways through the cell; a form that can be easily implemented in a variety of manners, including the Prolog approach that was taken.

This modelling system has converted the complicated three space environment into a network or graph, a construct that is easy to visualize, store, manipulate and search. The *nodes* of the network represent the doorways between the cells and the fixed points that the vehicle must pass through. The *arcs* represent the links between the doorways and capture the connectivity of structure and the free space. A path through the structure can be represented as a path from the starting position to the closest appropriate doorway, a sequence of paths between doorways and a path from the final doorway to the goal position.

A better understanding and a clearer picture of this modelling system can be obtained through example, as contained in the following sections. There are more details that must be considered before a final, workable model is obtained; these will be considered following these examples.

3.3.1 A Modelling Example in Two Dimensions

Although the modelling technique presented above was expressly developed to solve a three dimensional problem, its primary features can be demonstrated more clearly in two space, especially when drawings are involved. Figure 3.1 shows the steps involved in developing a model of a two space environment, as might be used by a 'floorwalking' style mobile robot. The environment contains obstacles that are typical of a slice taken through a platform structure, representing the end and side views of the tubing members.

In figure 3.1 (a) the initial environment with the obstacles is shown. The start and goal points (labelled S and G) of a trajectory to be planned are also shown. This example has been chosen to deliberately keep the number of doorways and links artificially small in an effort to avoid cluttering the diagrams.

In figure 3.1 (b) the workspace has been divided into convex polygonal cells that describe the free space. The cells are labelled A through D in the diagram. In most cases there will be a variety of ways to divide the environment at this stage. In general the number of cells and doorways should be kept as low as possible to obtain a compact model. In this example, as in the three space case, the boundaries of the cells are generally formed by portions of the obstacles.

In figure 3.1 (c) the doorways between the cells have been defined and labelled a through g. In the example all the doorways were assumed to be large enough for the vehicle to pass through. The doorways that link the section of the workspace under consideration to other regions are also shown.

Figure 3.1 (d) shows the equivalent network description that could be implemented and used for path searching. The model now consists of nodes and all the possible links between them. Note that links must be added between the start/goal points and the appropriate nodes to add them to the model. The structure of the network could be represented by information as shown in figure 3.2 which contains a Prolog type representation similar to the one that was implemented.

Finally figure 3.1 (e) shows a path through the modelled environment as would be generated by the planning system. The trajectory consists of a path from the starting point to the first doorway, a series of paths between doorways and a final segment from the last doorway to the goal point. By moving through the doorways that link the cells of free space the obstacles have been given a wide clearance, as is desired.

3.3.2 A Modelling Example in Three Dimensions

The following discussion presents a very simple example of the modelling technique applied in three space. The complexity of the example is limited since the figures are restricted to black and white drawings which rapidly become cluttered and unreadable for even a modestly complex workspace.

Figure 3.3 shows a portion of a structure similar to an offshore oil platform. The diagonal bracing members are shown on all the foreground planes but are omitted in the background for clarity, unless specifically required for the discussion.

Some terminology that will aid in this description, and will be used in later sections, will first be defined. The structure can be divided into four cubes, described by the main orthogonal bracing members, that have faces in three planar directions. The top and bottom of each cube are defined by structural members lying in *layer* planes. Similarly the front and back lie in *bent* planes and the ends of the cubes are in *row* planes. The structure shown in the figure contains three layer planes, two bent planes and three row planes. The generation of a model of this structure can now be discussed.

The first step that must be taken is to break the structure into the convex polyhedral cells of free space. This is straightforward in this case as the four cubes completely describe the contained free space. This is possible as there are no diagonal members that pass through the central region of the cubes, as is usually the case in full size offshore structures.

Figure 3.1: Two Space Model Development

The node (doorway) position information:

`position(a, [xa, ya]).`

`position(b, [xb, yb]).`

`⋮`

`position(f, [xf, yf]).`

`position(g, [xg, yg]).`

The position of the start and goal nodes:

`position(start, [xstart, ystart]).`

`position(goal, [xgoal, ygoal]).`

The links joining the doorways:

`link(a, c).`

`link(start, a).`

`link(start, b).`

`link(b, c).`

`⋮`

`link(e, f).`

`link(e, g).`

`link(e, goal).`

`link(goal, g).`

Figure 3.2: Network Facts for Two Space Model

Having chosen the cells, the next step is to define the doorways between them. Four doorways are indicated and labelled in the figure. Doorway $\mathcal{A}$ is located in the upper layer plane at the centroid of the bracing triangle around it. Doorway $\mathcal{B}$ is located in the centre row plane and joins the upper two cells (cubes). Again it is located at the centroid of the surrounding bracing triangle. Doorway $\mathcal{C}$ joins the two right hand cells and is located in the centre layer plane. The final doorway indicated: $\mathcal{D}$ is in the rightmost row plane. In general there are doorways located at the centroids of all the bracing triangles in the layer, bent and row planes. In the figure there would be a total of twelve doorways associated with each of the cell cubes.

Once the doorways have been defined they can be used as the nodes of the network model and the links can be defined. There are links from each node in a cell to all other nodes that are on a different face of the same cell. For the upper left cell that contains node $\mathcal{A}$, node $\mathcal{A}$ is connected to ten other nodes, including node $\mathcal{B}$. There would be 60 links associated with each polyhedral cell in the structure if there are diagonal braces on all the faces of the cubes. Although this is a fairly large number, it is not excessive, it only grows linearly with the number of cells and the storage requirements for each link are small. The fact based network model would contain information in the same form as that shown earlier in figure 3.2.

The final phase is the use of the model for path planning. This is illustrated in figure 3.4 where a path through the structure from node $\mathcal{A}$ to node $\mathcal{D}$ is highlighted. The path shown is not the shortest path between these two points but was chosen as it can be shown more clearly.

Figure 3.3: Three Space Modelling Example

Figure 3.4: Three Space Trajectory Example

3.4 More Modelling Details

Now that the basics of the modelling system have been described some of the finer details that were glossed over earlier can be considered. It is these factors that ensure that the modelling concepts can be incorporated into a viable world model, suitable for use by a path planner.

3.4.1 The Start and Goal Points

These points are the starting and finishing positions for the trajectory that will be planned by the navigation system, and are passed from the command computer. For a general purpose planner these points must be completely arbitrary, representing any position within the platform structure or in its immediate vicinity. The modelling technique discussed so far has no mechanism to add an arbitrary point to the world model network.

If the point in question is assumed to be in free space, a reasonable assumption considering the vehicle is endeavouring to get there, then it must be contained within one of the convex polyhedral cells of the model. If the correct cell is determined then temporary links can be added to the model that link the point to the appropriate doorways leaving the cell. The point has now become a temporary node in the network model and can now be treated the same as any other doorway by the planning system.

The problem of deciding which cell contains an arbitrary point requires a considerable amount of effort. In the work by Crowley [12,13] this problem was avoided by defining 'known places' beforehand and adding them to the model permanently. However this approach would be a severe limitation for a general purpose inspection vehicle so a solution was developed.

Additional information must be added to the world model to define the boundaries of each convex polyhedral cell that is not required for path planning. The

following rule is used to determine whether a cell contains a specified point:

- a point is within a cell if it is on the 'correct' side of all the planes that define the boundary of the cell.

The information required to utilize this rule was added in the following manner to retain flexibility and extensibility. Each cell is described by a list of the faces that make up its boundary and an arbitrary interior point. The face information includes a label, coordinates of a normal vector to the plane of the face, and the offset of the plane from the origin. This information is used to implement the rule test presented above using a procedure that performs a simple exhaustive search of the model cells, coupled with a quick pruning heuristic that rejects a cell as soon as the candidate point is on the wrong side of a face plane.

- 1 consider each polyhedron and continue until a cell is found that satisfies the above rule. If no cell is found then announce failure.
- 2 get the face information for the cell under consideration. Consider each face and test the point in the following manner.
 - 2a determine whether the interior point is above or below the plane containing the face.
 - 2b determine whether the input point is above or below the plane containing the face.
 - 2c If both points are on same side of the plane then test the next face of the cell. If all faces have been tested then the correct cell has been found. Otherwise try the next cell.
- 3 If a cell meeting all the face criteria has been found, announce success; otherwise announce failure.

This approach allows arbitrary points to be added to the model to represent the starting and goal points of the planned path. It also provides a mechanism through which new doorways could be dynamically added to the model within a cell. This would constrain the paths through the cell to pass through this point, a feature that could allow small obstacles to be added to the model during operation.

3.4.2 Link Scaling

In the model presented thus far the links between the doorways have all had the same properties. In general this is not the case in the workspace and it is advantageous to add a measure of the 'hazard factor' associated with traversing each particular link. This path weighting is added as a scaling factor that is used when the length of the particular link is computed. Paths that pass through safe regions of the platform are given a scaling factor of unity. Paths that pass through more hazardous regions that are to be avoided if possible, can be assigned a higher scaling factor. This makes the path *appear* longer to the path planner, thus less desirable. By suitably grading this scaling factor the path chosen through the structure can be optimized for path length and hazard.

The apparent length of the link (the actual length times the hazard factor) is calculated by the planner when a path is considered for traversal, rather than at the beginning of the mission. This approach allows for dynamic updating of the hazard factors in response to new information discovered during the mission. This provides a simple, reliable mechanism for adding information to the world model which enables 'fine tuning' of the model by the vehicle during operation.

If further control over the desirability of a specific link is required then the link scaling factor can be expanded. Two scale factors could easily be added if different factors were necessary for traversing the link in opposite directions, as might be the case in areas of high currents. Additional information could also be associated with each node (doorway) in the model that could be used in the path planning process.

3.5 Automatic Generation of the Model

Having determined the information that must be contained in the world model the next question to be addressed is how to generate it. There are two basic

approaches that can be taken: create the model information directly or generate the model data from information of the physical platform structure. The latter tack was taken as it is easier to create raw information that describes the platform structure itself rather than the direct free space description. The method used is outlined below.

3.5.1 Platform Definition

The first form of the model is a description of the members of the jacket platform. The physical location of all the platform joints were stored. Additional fields can be added at this stage to describe the joint in more detail. This information could include such things as the type of joint, tubing diameters, vehicle attachment points or cleaning information for the toolset system. The connectivity information lists all the tubing members of the structure that join the joints together. Again more fields can be added that contain additional pertinent information about the member such as the size or location of special features like corrosion anodes.

The information above describes the physical layout of the platform but gives no indication on how the free space description is to be developed. Extra information on the conceptual breakdown of the structure is necessary before the physical description can be converted into the final world model. A definition of the boundaries of the convex polygonal cells is added to the above data. Each cell is described by faces that make up its boundary and an interior point. The faces are described by the platform joints that lie in the plane of the face. This represents sufficient information to determine the doorways and the links between them.

3.5.2 Model Generation

The approach taken to convert the platform description to the internal world model is fairly straightforward as presently implemented. The doorways between the cells are calculated first. Each face is considered in turn and using the joint and connectivity data all the triangular openings within the face are found. If the opening is large enough the centroid is calculated and a node is then added to the network model at this position. Only triangles are used at the present time as all the openings are this shape due to the bracing scheme of the platform. This is typical of most offshore platforms so this limitation does not represent a severe shortcoming. The addition of a search for doorways of other shapes could be added if required.

Once all the nodes have been found the connectivity between them can be determined. Each convex polyhedral cell is considered in turn and all the links between the doorways are calculated. Care must be taken to connect only those nodes that are on different faces of the cell as there is no limit to the number of doorways that can occur on a single cell face.

The final stage of the model generation is to determine the geometric descriptions of the planes that contains each of the cell faces. The normal vector is calculated from the vector cross product of two of the edges of the face. The offset of the plane, from the coordinate system origin, is then calculated from the dot product of a point on the face and the normal vector.

All the data required for the free space world model in network form has now been generated. Further implementation details are contained in chapter 5.

3.6 Updating the Model and Machine Learning

The ability to create the model dynamically from data describing the platform allows for easier updating of the world model. For the present system the free

space network model is calculated completely before use by the planner. The system could be operated by calculating the model during the path planning phase as it is required by the navigation system. This feature would permit the vehicle, if equipped with the appropriate sensors and signal processing, to add data to the model if new features in the environment were discovered. A machine that learns as it goes, or more feasibly updates its knowledge, could then be developed. Whether this feature is necessary, feasible or even desirable for an underwater inspection vehicle has yet to be determined. Creating the model dynamically, as required, would reduce the storage requirements for the world model; but at the cost of increasing the time required to plan a suitable path. With the increased onboard memory that is currently available (and cost-effective) the precomputed case may well represent the better choice.

3.7 Discussion

This chapter has presented the ~~w~~orld modelling system that was developed at the conceptual level. The manner in which it was implemented was also described without getting into machine and language level details, which will be discussed in some detail in chapter 5, once the path planning system has been presented.

The modelling system developed represents a specific solution to the problem of representing an offshore oil platform for use in computer navigation. However, the approach is well suited to a much wider class of *three-dimensional* modelling problems for different applications.

The automatic generation of the model for the platform structure case saves considerable work in the compilation of the model data. There is no doubt that a much more sophisticated approach could be taken to this aspect of the problem that could further reduce the raw data input. The ultimate case would be a system that would generate the model directly from the engineering drawings (such as

those contained in appendix B) or from CAD/CAM data as it becomes available. This aspect of the problem was not stressed in this research, its development stemmed more from an attempt to save the author time and effort in preparing the data files required rather than a concerted design effort.

Chapter 4

Path Planning

This chapter discusses the path planning portion of the navigation system for an autonomous vehicle. The planner that was developed to use the world modelling technique discussed earlier will be presented and demonstrated.

4.1 Requirements of the Planner

The planner is responsible for finding a safe path between arbitrary start and goal points within the structure using the information contained in the world model. The path should minimize the distance the vehicle must travel but should also consider other hazard factors that could make a longer path more desirable. The output of the planner must be in a form readily usable by the other vehicle systems and should be generated in the minimum time possible.

4.2 Planning Techniques

In most mobile robot planning systems the design of the planner is strongly influenced by the modelling method employed. The design of the planner is made much easier if the model information is structured in a manner that lends itself to formal searching methods. A common form for the model, which permits the

use of a wide variety of search methods, is a network or graph. All the modelling methods mentioned in the previous chapter had a final network structure of nodes with links between them in some form.

There are many methods that can be used to search for a specific path through a network or graph. The most suitable techniques usually involve a searching strategy that advances several paths in parallel and further develops the most promising candidate according to some weighting scheme or heuristic until the goal is reached or all the paths have been tried. The most common technique is to utilize a form of the A* searching method developed by Nilsson [38]. The A* method is an extremely powerful heuristic graph searching method well suited to large complex networks.

The A* method weights each path under consideration by the following function:

$$f(n) = g(n) + h(n)$$

where $f(n)$ is an estimate of the cost of the path to the goal constrained to pass through node n , $g(n)$ is the cost of the path from the starting node to node n and $h(n)$ is the *heuristic function* that provides an estimate of the cost of the path from n to the goal node. If $h(n)$ is a lower bound estimate of the actual cost of the optimal path from n to the goal then it can be shown that the search algorithm is *admissible*. This implies that if there is a path from the start to the goal the A* search will terminate by finding an optimal path between them. The nature of the function $h(n)$ determines the effectiveness of the search: the closer $h(n)$ estimates the actual cost of the remaining path the faster the search will converge to that path.

4.3 Path Planner Design

The design of the path planner that was developed to work in conjunction with the world modelling technique presented earlier was greatly simplified by the appropriate form of the model. By using a network form for the model the A* style of graph searching algorithm could be employed.

The characteristics of the graph searching method are controlled by the choice of the $g(n)$ and $h(n)$ functions. It was found that significant improvements in the performance of the search algorithm could be realized through altering these functions. The design of the $g(n)$ and $h(n)$ functions will be outlined in the following sections.

4.3.1 The $g(n)$ Function

The function $g(n)$ represents the cost of the path developed so far between the start node and node n . This was taken as the sum of all the individual lengths of the links traversed to get to node n from the start point. Each individual link distance is scaled by its hazard (or scaling) factor that was discussed earlier. The form of the $g(n)$ function is thus as follows:

$$g(n) = \sum_{i=1}^n \text{scale factor}_i \times \text{link length}_i$$

The scale factor is taken as unity for normal safe links and is increased accordingly for more hazardous links. The factor cannot be made less than one without compromising the accuracy of the $h(n)$ estimator. The scaling factors must be used with discretion as the $h(n)$ estimator becomes increasingly less accurate as the number of links with scale factors greater than one is increased.

It is through the use of the link scaling or hazard factor that the nature of the 'optimal' path can be controlled. The definition of what is considered an optimal path for an offshore vehicle is fairly subjective and may be influenced by

a multitude of factors depending on the type of vehicle, the environment and the nature of the mission being performed. In most cases the actual distance that the vehicle must travel to get from one point to another should be minimized, to conserve onboard resources and reduce mission time. However if the shortest path must pass through a dangerous portion of the workspace there may be situations where a little extra distance travelled is worth the expense. If the detour must be large then the best solution would be to pass through the more dangerous region regardless. This approach is much more flexible than a 'black and white' system consisting of only free space or obstacles.

The link scaling factor can also incorporate other factors. Short links that would tend to give a more complex path could be given higher scaling factors. The ability to influence the path that is chosen through such a simple means makes dynamic alteration of the scaling factors a fast and reliable method a fine tuning the model. This separation of the structure or connectivity of the model and the merits of each individual path produces an extremely flexible and advanced path planning system.

It should be noted that the A* search guarantees an optimal path in the form of the shortest distance between the start and goal points using the actual lengths assigned to each arc in the network.

4.3.2 The $h(n)$ Function

The heuristic function $h(n)$ was based on the actual straight line distance from the node at the head of the partial path to the goal node. This distance must be a lower bound estimate of the actual distance from a node to the goal. This distance is then multiplied by a *search factor* (sf) that controls the nature of the search. The $h(n)$ function thus becomes:

$$h(n) = \text{search factor} \times \text{distance to go}$$

The number assigned to the search factor has a profound effect on the search strategy. If the search factor is set to less than one then the search will remain admissible and will always terminate on an optimal path. As the factor is reduced the search strategy becomes more 'breadth first' as the effect of the heuristic function is reduced. If the search factor is set to one then the best lower bound estimate of the distance remaining is used and the best A* performance is obtained.

If the search factor is increased from one then the lower bound guarantee on the $h(n)$ function is no longer valid, an element of a 'depth first' search strategy is added and the search is no longer of type A*. The search strategy is now classified as a type of *branch and bound* search with *dynamic programming* to remove loops and a *forward estimator* of the distance remaining [52]. In this case more weight is given to a partial path whose tip node is closer to the goal rather than a shorter path that has not progressed as far towards the goal. This tends to reduce the number of partial paths that are considered and will produce a faster solution providing the heuristic function is relatively accurate. However there is no longer a guarantee that the first path found by the planner will be the shortest route between the start and goal nodes.

In many problems the requirement to find the *best* path is less important than finding an *acceptable* path within a shorter period of time. Due to the connectivity of the free space in the offshore oil platform model there tends to be a large number of alternate paths between any two points connected by a reasonably large distance. In general many of these paths have lengths that differ by only a small amount. If the shortest of a large set must be found (as with any admissible search method) then a significant effort must be expended to choose within the set. By assigning the scale factor to a number greater than one only the best of the set of similar paths tend to be considered for expansion rather than the entire set. For the offshore oil platform model a decrease of the search time of more than an order of magnitude was realized using scale factors between 1.1 and 1.3. In all

cases the path that was found was within 5% of the length of the optimal path, a small penalty for the dramatic performance increase. Detailed examples of the search parameters and the planner performance are given in the following chapter.

4.4 The Nature of the Path

The output of the planner is a description of the optimal path between the start and goal points. The path is described in the form of a series of waypoints through which the vehicle must pass as it progresses along the path. The individual path segments between the waypoints are straight line segments in three space. This path gives the obstacles a wide berth, although at the expense of a somewhat longer path in most cases. This is demonstrated in the paths shown in the modelling examples presented in the previous chapter: figures 3.1 and 3.4.

The extra path length is generally a small price to pay for the added security. Since the path is well clear of obstacles it does not necessarily have to be followed with great precision, a feature that can simplify the local navigation and vehicle control portions of the onboard computer system. However methods could be developed that would smooth out some of the portions of the path, although none of these methods were incorporated into the system that was implemented.

4.4.1 Optimization

The simplest form of path optimization would be to move the doorways from the centroid of the bracing triangles. Each node in the network model could incorporate a more detailed description of the doorway including the size of the 'window' in the bracing members. After the path had been chosen the waypoint associated with the doorway could be moved closer to the edge of the 'window' to smooth out the path. This path is still guaranteed to be safe as the revised doorway position is still on the surface of the free space cell.

Relaxation techniques can be used to determine the best location of the doorways to smooth out the path. This method has been used successfully in two dimensions by Thorpe and Matthies [33,49]. In this approach the position of each doorway is adjusted to minimize the cost of each of the local path segments. This procedure must be repeated as moving one node will affect the cost of the neighbouring path segments. Each node is repeatedly revised until no nodes move more than some small amount. This approach would give a reasonable method of smoothing out the final path between the start and goal points and reducing the total path length.

4.5 Planning Issues

An alternative to planning a complete path for each route between two points would be to use a suite of preplanned routes between each pair of cells. This would reduce the planning problem to a search for the best of these routes. This approach could give faster performance at the price of reduced generality. The preplanned route solution was rejected as it was felt that the reduction in planner generality and functionality was too steep a price to pay for the increase in performance, given that acceptable real-time performance can be obtained using a powerful planner to solve the general problem. In addition the problem of start/goal point classification must still be solved and the number of preplanned paths required rises rapidly as the number of model cells increases.

The manner in which the planner is used should also be considered. If a significant amount of path replanning is anticipated (eg. due to change in mission requirements) then it is possible that a more useful global navigation system could be developed using a planner that planned paths from the goal to the starting point that were then executed in reverse. The potential improvement in functionality arised from the ability to reuse some of the partial paths that were developed

(and added to the database) in the replanning process as those alternatives close to the goal would not be affected by a 'new' start point. The distance to the 'goal' would have to be revised in the replanning case but this should take less time than recomputing all relevant paths. This feature was not investigated further as the focus of the system that was developed was towards world modelling and initial path planning.

4.6 Discussion

This section has described concepts involved in the path planning portion of the navigation system. A method that finds an acceptable path within as short a period of time has been presented. Specific examples of planned paths were given in the previous chapter during the modelling examples. A further example of the planner will be provided in the following chapter along with more details of the actual implementation of the planning and modelling methods.

Again it must be stressed that the apparent simplicity of the planning portion of the navigation system is due to the appropriate choice of modelling technique which converts the complex workspace into a simple, regular and flexible format.

Chapter 5

Prototype System

The modelling and planning methods described above have been implemented using the MProlog language [29,30] which supports modular Prolog program development. This language was chosen as it allows rapid prototyping, provides an extensive program development environment and lends itself to the fact based, network modelling technique and heuristic search procedure used to find paths through the structure.

MProlog is available for a wide range of mainframe and minicomputers for program development. There are also versions ported to a variety of personal computers, a feature that allows the planning and modelling software to be run on hardware similar to that suitable for an autonomous vehicle. More details of the Prolog language can be found in the references [8,9].

The modelling and planning software was developed and tested on a Vax 11/750 computer running the Unix¹ operating system. The final software was also transferred to an Atari 1040ST personal computer for further performance testing. The personal computer environment more closely represents the hardware that would be used for an autonomous underwater vehicle.

This section covers the implementation of the prototype system and considers the capabilities and performance of the modelling and planning systems devel-

¹Unix is a registered trademark of Bell Labs.

oped. The Prolog source code for the majority of the prototype implementation is contained in Appendix A.

5.1 Path Planner Implementation

The path planner searches the world model using the searching algorithm presented earlier. Each partial path that is being advanced is represented by a dynamic assertion containing a list of the paths nodes, its length and its merit value ($f(n)$). This approach was found to be considerably faster than using a more formal approach where the partial paths were manipulated in a single list. A test was incorporated to remove any partial paths containing loops as they were generated. A pruning heuristic was used to remove any partial path that reached a node that had already been reached in a shorter distance. Using these techniques the final version of the planner runs much (roughly two orders of magnitude) faster than the initial implementation.

The length of each link is calculated as the path is expanded using the calculated link length times the appropriate link scaling factor, rather than being precomputed. The heuristic function is also calculated as the link is expanded, using the latest version of the search factor. Both these features allow the model and search characteristics to be dynamically changed during the operation of the planning system. This adds considerable flexibility to the system, a feature that would be required for a vehicle performing long, complex missions.

5.2 Performance

The following performance figures are quoted for the Prolog prototype system running on a VAX 11/750 computer under the Unix operating system. The navigation module has been tested on a variety of world models. The most extensive test was

using the test platform structure that is shown in figure 5.1. This network model of this structure contains 18 polyhedral cells, 216 doorways and almost 2000 links. This model contained most of the features found in a typical offshore oil platform such as that shown in the drawings contained in Appendix B.

The time taken to plan a path through this structure is strongly dependent on the search factor that is chosen. It was found that a search factors in the range of 1.1 to 1.3 gave significant performance increases over a factor of 1 (as required for an A* search) and still found a path within a few percent of the optimal (often still the optimum path). Using a factor greater than this did not produce a further significant decrease in the search time. The performance of the system is demonstrated for two of the modelling cases that were tested. A variety of runs, demonstrating different features of the system, will be presented.

The following performance figures pertain to the large test platform shown in figure 5.1. The trajectories are not shown on the drawing as they would be difficult to illustrate clearly. Planning a path from one corner of the structure to the other, the worst case condition, which contained eight links was accomplished in 55 seconds using a search factor of 1.2. Only nine partial paths were considered during the planning process. The output of this run is shown in figure 5.2. This should be contrasted with the same path using a search factor of 1 that took over 11 minutes, after considering 102 paths. In this case the optimal path has been found at the expense of greatly increased run time. The output for this run is shown in figure 5.3 for comparison with the previous example. A shorter path of five links was found in 29 seconds after considering only 5 paths. The program output contains the scaling factor, the start and goal point coordinates, the CPU time used to plan the path, the path shown as a series of waypoints denoted by the internal model label and the coordinates, the scaled length of the path and the number of node expansions that were performed during the path planning.

The following performance figures pertain to a simpler structure and are pre-

Figure 5.1: The Large Test Platform Structure

Enter the scaling factor

*1.2e0

Enter the start position coordinates

*5

*5

*5

Enter the goal position coordinates

*25

*5

*55

cpu time to find path: 55 secs.

Node	X	Y	Z
start	5.0	5.0	5.0
12	6.67	8.33	10.0
159	10.0	8.33	16.67
23	13.33	5.0	20.0
33	16.67	8.33	30.0
42	16.67	8.33	40.0
186	20.0	8.33	46.67
53	23.33	5.0	50.0
goal	25.0	5.0	55.0

The path length is : 59.2

The number of paths considered was: 9

Figure 5.2: Program Run for Large Test Structure, Search Factor of 1.2


```
Enter the scaling factor
*1
Enter the start position coordinates
*5
*5
*5
Enter the goal position coordinates
*25
*5
*55
cpu time to find path: 662 secs.
Node      X      Y      Z
start     5.0     5.0     5.0
12        6.67    8.33    10.0
159       10.0    8.33    16.67
23        13.33    5.0     20.0
31        16.67    1.67    30.0
182       20.0    1.67    36.67
44        23.33    5.0     40.0
53        23.33    5.0     50.0
goal      25.0    5.0     55.0

The path length is : 59.2
The number of paths considered was: 102
```

Figure 5.3: Program Run for Large Test Structure, A* Search (Search Factor of 1)

Figure 5.4: Small Test Structure, showing Simple Path

sented to demonstrate the effect of the link scaling factor. Figure 5.4 shows the simple structure and the normal shortest path between the centres of the upper left and upper right hand cubes of the structure. The program output showing the planning of this path is shown in figure 5.5.

In figure 5.6 the upper middle cube has been deemed dangerous for some reason and all links that pass through it are assigned a scaling factor of 2, effectively making them twice as long as far as the planner is concerned. The resulting 'shorter' path through the bottom of the structure has been shown on the diagram. The output of the planner for this path is shown in figure 5.7.

These examples have been chosen to demonstrate the operation and capabilities of the modelling and planning system that was implemented. The system has been tested on an assortment of model structures, planning simple and complex paths.

```
Enter the scaling factor
*1.2e0
Enter the start position coordinates
*5
*5
*5
Enter the goal position coordinates
*25
*5
*5
cpu time to find path: 4 secs.
Node      X      Y      Z
start     5.0     5.0     5.0
34        10.0    6.67    6.67
38        20.0    6.67    6.67
goal      25.0     5.0     5.0

The path length is : 21.6
The number of paths considered was: 3
```

Figure 5.5: Program Run for Figure 5.4

Figure 5.6: Small Test Structure, showing Complex Path

```
Enter the scaling factor
*1.3e0
Enter the start position coordinates
*5
*5
*5
Enter the goal position coordinates
*25
*5
*5
cpu time to find path: 24 secs.
Node      .X      Y      Z
start     5.0     5.0     5.0
8         6.67    6.67    10.0
47        10.0    3.33    13.33
51        20.0    3.33    13.33
11        23.33   3.33    10.0
goal      25.0    5.0     5.0

The path length is : 31.54
The number of paths considered was: 14
```

Figure 5.7: Program Run for Figure 5.6

Starting to make the network model.
Time taken to find the nodes was: 103 secs.
Time taken to find the links was: 191 secs..
Time taken to find the face planes was: 5 secs.
Finished writing model

Figure 5.8: Model Generation Output for Large Test Platform

The system is capable of handling large models without rapid growth in the run times for the model generation and path planning.

The time taken to generate the model from a description of the platforms geometry is roughly proportional to the size of the model. The time required to generate the network model for the large test platform was about 5 minutes, split up as shown in figure 5.8.

5.2.1 Planning Time

The time required to plan a path is more dependent on the size of the model. There are two factors that increase the time taken to plan a path for a large model. The size of the model increases the time taken to search through the Prolog database for the appropriate facts. This factor increases roughly linearly with the size of the model, but represents only a portion of the path planning execution time.

The second factor is the larger number of alternate paths that must be considered when planning a long path in a large, complex model. This was found to be a much more significant factor in determining the time required to plan a path. This factor can only be controlled using a powerful search procedure. The powerful search strategy using search factors in the range of 1.1 to 1.3 provides a focused search that limits the number of alternatives that must be considered compared to an A* type of search. A search factor of 1.2 was found to be a suit-

able value for most planning cases. As a result of these factors the time required to find a path is a relatively weak function of the size of the model and increases geometrically (roughly linear) with the length of the path being planned.

It should be noted that theoretically there is a potential exponential growth in the planning time as the model gets larger. This rate of growth was not observed due to the nature of the physical environment where a path can always be found between two points within the structure. If the connectivity of the free space was dramatically reduced then the planning system would need reevaluation.

5.3 Personal Computer Performance

The code developed and tested on the Vax computer was also transferred to an Atari 1040ST personal computer for further testing. This computer uses a Motorola MC68000 microprocessor chip and has one Megabyte of memory. This environment is representative of the hardware that is used on current autonomous underwater vehicles. The operation in this environment should be a good indication of the performance in an actual vehicle.

The program for the Atari required no major changes from the Vax version. The only changes related to the use of external language predicates that were not supported in the PC version. The external language predicates used in the Vax version were re-coded in Prolog.

The same test cases were run on the Atari that were used for the Vax. In general there were no major differences in performance observed for the same program running on the Vax or Atari computer. For example, the time taken for the same run as shown in figure 5.2 was 58 seconds on the Atari compared to the 55 seconds for the Vax. For long program runs (in excess of five minutes) the Atari version started to take slightly longer. For all the test cases the performance decrease for the Atari environment was under 20 percent. For very long runs, such

as is the case with a poorly chosen search factor, the Atari computer appeared to run out of memory ². Providing extra memory in an operational environment would not be difficult.

In general the operation and performance of the planner and world modelling program in the micro-processor environment was definitely adequate for an operational vehicle.

5.4 Prototype System Code Overview

This section contains a brief description of the major Prolog code sections that are contained in appendix A. This is not a description of how the code works but rather a summary of how the modules fit together. The data files and world model for the example structure shown in figure 5.6 are used for demonstration in the appendix.

5.4.1 Implementation Approaches

Two approaches were taken to the prototype system, differing chiefly in the dynamic aspects of the implementation and the updating of the model that was possible. The static approach used a world model that was represented entirely by static Prolog facts (or assertions) that could not be changed during the operation of the planner. The dynamic alternative used a world model consisting of dynamic assertions that could be created, destroyed or changed as the program was running. The two approaches can also be combined to create a graded dynamic system where limited updating of the model is possible during operation.

The static implementation uses a world model that is represented completely by Prolog source code. This information is then used directly by the planner to

²The Atari version of MProlog was a beta test release so the failure of the long run could have been due to other factors.

generate paths through the structure. Temporary information is added to handle the arbitrary start and goal positions during the planning process. The model can be generated automatically from a description of the platform structure by a separate program, the output of which is the Prolog source code that represents the links and nodes of the network world model. This represents the simplest approach to the navigation module.

In the dynamic implementation the world model is created dynamically at the start of operation and consists entirely of dynamic assertions added to the Prolog database. These facts can then be manipulated during the remaining program operation. The links and nodes associated with incorporating the start and goal positions can be added to this information. The link scaling factors can be updated during operation and link lengths are computed during program operation. There is sufficient flexibility in this approach to enable the incorporation of machine learning features as required.

The dynamic aspects of the implementation were not used to any great extent in the prototypes developed, mainly as the model updating information would come from a different portion of the vehicle system and would be dependent on the sensing capability of the vehicle. The dynamic approach was implemented primarily to demonstrate the feasibility of the method and the power and flexibility of the modelling paradigm that was developed.

The graded system could incorporate features of both these approaches, allowing simple model updating through the use of temporary additions to the model or through updates to the link scaling factors or the actual link length.

5.4.2 The World Model

The world model was implemented by a series of assertions to represent the nodes of the model and the arcs between them. Additional facts were added to capture the conceptual model of the platform in sufficient detail to allow the planner to add

additional points to the network model. The network world model for the example structure depicted in the figure 5.6 is shown in module `scaled_demo_model` in the appendix to demonstrate the fact structure used to capture the features of the world model.

There are two fact types in the `scaled_demo_model` module; the `model_node` facts that describe the nodes (doorways) in the model and the `model_link` facts that describe the connectivity.

The world model can be constructed from a description of the platforms geometry and a description of the cell that it should be divided into. The code in module `model_builder` takes a description of these facts and generates facts of the form found in the module `scaled_demo_model`. The description of the platform geometry and connectivity for the example case is shown in the module `demo_plat`. The facts that describe the corresponding cells of free space are shown in the module `demo_cells`.

5.4.3 The Path Planner

The Prolog code for the path planning system is contained in the module `astar5`. This code uses the facts in the network model such as `scaled_demo_model` and the description of the cells of free space, as in `demo_cells`, to plan an optimal path between two points specified by the user. The operation of the planner falls into two phases, the first where the start and goal points are specified and subsequently added to the world model and the second where the path from the start to the goal is planned. A brief description of the steps involved is given in the following paragraphs.

The path finding process is controlled by the predicate `path_finder`, which cleans up all dynamic data from the previous path, calls for the input data, sets up the appropriate dynamic assertions, finds the path and then displays the results. During the planning process all the information that is generated towards finding

the appropriate path is stored in the form of dynamic assertions to the Prolog database. This method was found to offer significant performance improvements over more traditional list or array based data manipulations.

There are four groups of dynamic assertions that are used during the path planning process. Each `partial_path` assertion represents a partial path that is being advanced during the path planning process. The open assertions keep track of all the nodes that have been expanded and the lowest cost path to the node that has been found to date. The `temp_node` assertions add the nodes associated with the start and goal points to the world model. The `temp_link` assertions are used to connect the start and goal nodes to the other adjacent nodes in the model.

The `read_start` and `read_goal` predicates read in the coordinates of the start and goal points, add the temporary node to the world model and then search for the cell that the point is contained in using the `find_poly` predicate. The `find_poly` predicate searches through the cells and performs the plane normal test discussed in the design section. Once the cell that contains the start/goal point has been found then the appropriate assertions can be added to the model to connect the point to the adjacent nodes in the model.

It is the `expand` predicate that actually finds the path from the start to the goal. The first clause tests the `partial_path` facts to determine whether the goal has been reached and ensures that the possibility of a better path does not exist by the `better_solution` predicate. The second clause is used if an acceptable path has not been found. The `select` predicate finds the best partial path to expand and the `generate_alternatives` predicate expands the node at the head of the partial path chosen and adds the new `partial_path` assertions to the top of the Prolog database.

The precise details of how each predicate operates and is used can only be determined by studying the Prolog code found in the appendix.

5.4.4 Other Modules

There are two other modules shown in the appendix. The module **controller** is used to coordinate the actions of the model builder and the path planner to dynamically generate the model and then use it for path planning. The module **utilities** contains some commonly used utility predicates that are used by the other modules.

5.4.5 Code Size

The total size of the binary code that can be run directly by the MProlog Interpreter³ is about 150 kbytes, of which about 100 kbytes is used by the world model database. The Interpreter program required to run this system is about 250 kbytes for the Atari PC version. The dynamic memory requirements for this program when planning a long, complicated path are in the 500 kbyte range. Thus the total size of the operational planner, world model, run time system and scratch memory is in the order of 1 Mbytes, well within the memory sizes that could easily be provided for an operational vehicle.

5.5 Integration into an Underwater Vehicle

The navigation system that was designed and implemented was not developed for any specific underwater vehicle. It was designed in a manner that should make its integration into a real vehicle possible and relatively straightforward. The design was aimed towards a distributed, multi-processor vehicle control system where the navigation system containing the world model is implemented on a separate processor. The information that is presently entered as input for the path planning system would be sent in a message from the vehicle control computer. When a

³For further information on the MProlog development environment see [29]

path had been found it would be sent as a reply to the control computer in the form of the *waypoints* (3D coordinates) that are printed by the prototype system. The remaining information could be sent as required by the rest of the system.

If some of the dynamic features that have been discussed were to be incorporated then additional interface procedures would have to be added to the navigation system. These could be fairly simple and would utilize the additional flexibility that is inherent in the modelling and planning system. These features would be initiated by other messages from the control computer or other control system processors.

The performance of the system should be sufficient for an underwater vehicle. Since the speed of the vehicle is slow and the distances to travel are fairly long a planning time of less than one minute is an adequate real-time response. A system designed to run on dedicated hardware for an underwater vehicle would be expected to run significantly faster than the prototype system. For example; experience with other MProlog programs running on a DEC micro-Vax or Sun 3 workstation has seen performance increases of 2 to 4 times over the same program running on the Vax 11-750 used for the prototype. The performance obtained from the development program running in the Atari PC environment further demonstrates the feasibility of using this system for an operational vehicle.

In general there is nothing in the design or implementation of the prototype system that should make its integration into a real underwater vehicle problematic.

Chapter 6

Conclusions

This thesis has presented a world modelling and path planning system that was developed to generate optimal trajectories between arbitrary points within a three dimensional environment. The method was developed bearing in mind the constraints of a 'real life' system and has been implemented and tested in an environment similar to that which would be found in an operational vehicle. The integrated modelling and planning system represents a flexible and extensible design framework that allows straightforward updating and enhancement. The key features of the method that has been developed and areas where additional research could be carried out are summarized in the following sections.

6.1 Results

The primary result of this research is the development of a workable navigation system for a large, three dimensional workspace: an offshore oil drilling platform. An integrated world model and path planner has demonstrated acceptable real-time performance when planning long, complex paths within a large offshore structure. The design of the navigation system was driven from a realistic computer architecture for an operational autonomous underwater vehicle to ensure simple, straightforward integration with other typical vehicle systems.

6.1.1 Modelling

The modelling system that has been developed provides a solution to the problems associated with trajectory planning within a three-dimensional workspace. The modelling technique effectively and efficiently captures the free space within a large offshore oil platform. The world model divides the free space within the workspace into large segments so that a large model can be captured without requiring an excessively large computer representation. In addition smaller cells of free space can easily be added in areas where increased resolution is required.

The world model was designed to be easily extensible so additional information pertaining to the operating environment could be added. The simple framework of the model makes adding and updating information straightforward. This information could also be added dynamically during the operation of the vehicle if appropriate sensors were available and an additional interface was provided.

The modelling system incorporates a distinct scaling factor that is associated with each link in the world model to indicate the desirability of a specific path segment. This allows for detailed grading of the paths within the structure based on arbitrary environmental factors. This scaling factor is utilized by the path planner when assessing the relative merit of a specific route between the start and goal points.

A rudimentary method of automatically generating a network world model from a description of the platform geometry and a framework of the cells of free space has been developed. This tool represents a first step towards the development of the world model from a higher level, obstacle based description of the workspace. However the generation of the world model is still a relatively manual process where the high level conceptual model must be provided directly rather than being automatically produced from the physical environment description.

6.1.2 Planning

An efficient planner that rapidly converges on an acceptable path through a large world model has been developed. This path represents an 'optimal' route between arbitrary points within the workspace based on a wide variety of physical and environmental parameters that can be captured in the link scaling factor. The ability to utilize this additional information pertaining to the workspace directly enables the planner to consider a wide variety of factors in the determination of the optimal shortest path.

The planner uses the dynamic programming features facilitated by the Prolog language to represent the state of the planning process in a flexible and easily accessible manner. Through optimization in the planner and the model acceptable real-time performance has been obtained when planning a long path with a large, complex structure. While the performance that has been obtained is sufficient for an underwater vehicle a further decrease in the planning time would further improve the usefulness of the navigation system.

The planner incorporates a method of adding arbitrary points to the model for use as start and goal points of a planned path. This feature allows for completely general path planning within the workspace. This is an important feature as it separates the model generation and utilization processes, there being no requirement to identify specific start or goal points during the model generation stage. The ability to add general points to the world model dynamically could also be utilized alter the model during a vehicle mission to reflect a changed workspace.

6.2 Extensions and Future Work

The system that was developed operates successfully as a prototype running in the Lab. The system was designed to be as general and extensible as possible. Thus there are many areas where additional features could be added to the mod-

elling and planning systems to provide a more capable navigation system for the vehicle. Some of the key areas where additional research and development would be interesting and beneficial are highlighted in the following paragraphs.

Areas of additional research from the navigation system point of view as a whole are:

- incorporate the navigation system into an underwater vehicle.
- investigation of the dynamic aspects of the system to support machine learning capabilities.
- adaptation of the modelling and planning method to other three-dimensional workspaces.

There are features of the modelling system that warrant further investigation in an effort to improve the features provided by the model. These include:

- enhance the capabilities of the model building system to capture information from a CAD/CAM design database.
- the automatic selection of the optimal set of free space cells for a model.
- automatic generation of the path weighting factors based on an assessment of the features of the workspace.

There are features of the path planning system that could also be investigated. These include:

- investigate methods of further improving the performance of the path planner, such as dynamic control of the search factor.
- addition of the path smoothing (relaxation) method to further optimize the planned path.

6.3 Contributions

The significant contributions of this work to the underwater vehicle and path planning fields are summarized below:

- an integrated modelling and planning method for a three-dimensional workspace.
- a modelling method suitable for representing large 3-D structures incorporating:
 - closely packed convex polyhedral cells.
 - separation of model structure and paths.
 - individual link scaling factors.
- an efficient path planner incorporating:
 - arbitrary start/goal points.
 - powerful search strategy

6.4 Conclusion

This system represents a solution to the particular navigation problems of an underwater vehicle operating within an offshore oil platform. However the approach used and the techniques developed are applicable to a wide range of mobile robot applications. Neither the modelling technique nor the path planning method are tied strongly to the underwater environment. This system could be easily adapted to many other machines required to operate autonomously in a three-dimensional workspace.

References

- [1] D. R. Blidberg, A. S. Westneat and R. W. Corell, "Expert Systems, a Tool for Autonomous Underwater Vehicles," *Proceedings of Trends & Applications, 1983: Automating Intelligent Behaviour, Applications & Frontiers*, Gaithersburg, Maryland, May 25-26, 1983, pp.149-209.
- [2] D. Richard Blidberg, "Time-Ordered Architecture for Knowledge-Based Guidance of an Unmanned Untethered Submersible," *Oceans 84 Conference Record*, Marine Technology Society, IEEE Ocean Engineering Society Washington DC, Sept. 10-12, 1984, pp. 571-575.
- [3] Rodney A. Brooks, "Solving the Find-path Problem by Good Representation of Free Space," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-13, No. 3, March/April 1983, pp. 190-197.
- [4] Rodney A. Brooks, "Planning Collision-Free Motions for Pick-and-Place Operations," *The Int'l Journal of Robotics Research*, Vol. 2, No. 4, Winter 1983, pp. 19-44.
- [5] Brown and Root International, Inc., "Platform Engineering Drawings", *Brown and Root International, Inc.*, Post Office Box 3, Houston, Texas. December 1985.
- [6] F. R. Busby and Associates, "Remotely Operated Vehicles," *US Department of Commerce*, No. 03-78-G03-0136, August, 1979.
- [7] Raja Chatila, "Path Planning and Environment Learning in a Mobile Robot System," *Proceedings of the European Conference on Artificial Intelligence*, Orsay, France, 1982, pp. 211-215.
- [8] K. L. Clark and F. G. McCabe, *micro-PROLOG: Programming in Logic*, Prentice-Hall, Englewood Cliffs, NJ, 1984.
- [9] W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, Springer-Verlag, New York, NY, 1984.

- [10] Robert D. Collier and Lawrence Goldberg, "Review of Non-destructive Evaluation Techniques and Their Application of Fatigue Cracks and Overstressing in Underwater Welded Steel Structures," *Oceans 81 Conference Record*, IEEE & Marine Technology Society, Boston, Ma., Sept. 16-18, 1981, pp. 447-453.
- [11] Tim A. Cornitius, "Diving Companies see More ROV Use," *Offshore*, February, 1984.
- [12] James L. Crowley, "Position Estimation for an Intelligent Mobile Robot," *The Robotics Institute 1983 Annual Research Review*, Robotics Institute, Carnegie-Mellon University, Pittsburg, Pa., 1984, pp. 41-50.
- [13] James L. Crowley, "Navigation for an Intelligent Mobile Robot," *Carnegie-Mellon Robotics Institute Technical Report 84-18*, The Laboratory for Household Robots, The Robotics Institute, Carnegie-Mellon University, Pittsburg, Pa., August 1984.
- [14] James L. Crowley, "Dynamic World Modelling for an Intelligent Mobile Robot," *Seventh International Conference on Pattern Recognition*, Montreal, Canada, August 1984, pp. 201-210.
- [15] Robin M. Dunbar, Stephen J. Roberts and Stuart C. Wells, "Communications, Bandwidth Reduction, and System Studies for a Tetherless Unmanned Submersible," *Oceans 81 Conference Record*, IEEE & Marine Technology Society, Boston, Ma., Sept. 16-18, 1981, pp. 127-131.
- [16] W. R. Edwards and D. M. G. Hargreaves, "Five Year Underwater Inspection Program of a North Sea Steel Jacket Platform," *Proceedings of the 11th. Annual Offshore Technology Conference*, Houston, Texas, April 30 - May 3, 1979, Vol I, pp. 234-248.
- [17] John F. Gilmore, "Automatic Route Planning in Autonomous Vehicles," *Seventh Int'l Conference on Pattern Recognition*, Montreal, Canada, August 1984, pp. 880-882.
- [18] W. J. Graff, *Introduction to Offshore Structures*, Gulf Publishing Co., Houston, Texas, 1981.
- [19] S.Y. Harmon, "Autonomous Robot Submersibles: The Future of Unmanned Submersibles," *Computers in Engineering 1982: Proceedings of the 2nd. Int'l Computer Engineering Conference*, San Diego, California, Aug 15-19, 1982, pp. 33-36.

- [20] Hudson Engineering Corporation, "Platform Engineering Drawings", *Hudson Engineering Corporation, a McDermott company*, 801 North Eldridge St., P.O. Box 218218, Houston, Texas. December 1985.
- [21] Lin Huiyong, "An Auxiliary Navigation System for an Autonomous Underwater Vehicle," *Oceans 84 Conference Record*, Marine Technology Society, IEEE Ocean Engineering Society. Washington DC, Sept. 10-12, 1984, pp. 604-609.
- [22] Yoshiaki Ichikawa and Norihoko Ozaki, "Autonomous Mobile Robot," *Journal of Robotic Systems*, Wiley, 1985, pp. 135-144.
- [23] James C. Jalbert, "EAVE-East - Goals and Applied Technologies," *Proceedings of the Third International Symposium on Unmanned Untethered Submersible Technology*, Marine Systems Engineering Laboratory, University of New Hampshire, Durham NH., June 6-9, 1983, pp. 62-76.
- [24] James C. Jalbert, "EAVE-East Field Test Results," *Oceans 84 Conference Record*, Marine Technology Society, IEEE Ocean Engineering Society Washington DC, Sept. 10-12, 1984, pp. 582-587.
- [25] James C. Jalbert, Carl N. Beverly and Steven G. Chappell, "EAVE-East Cavitation Cleaner Feasibility Demonstration," *Proceedings of ROV '85*, The Marine Technology Society, San Diego, Ca., April 2-4, 1985, pp. 215-221.
- [26] O. Kristiansen and G.I. Sletten, "Diverless Underwater Inspection Program," *Norwegian Maritime Research*, No. 1, 1984, pp. 23-29.
- [27] Darwin T. Kuan, Rodney A. Brooks, James C. Zamiska and Mangobinda Das, "Automatic Path Planning for a Mobile Robot Using a Mixed Representation of Free Space," *Proceedings of the First Int'l Conference on Artificial Intelligence Applications*, IEEE Computer Society, December 1984, pp. 70-74.
- [28] Griff C. Lee, "Recent Advances in Design and Construction of Deepwater Platforms-Part 1," *Ocean Industry*, November 1980, pp. 71-80.
- [29] Logicware Inc., *MProlog Language Reference, Release 2.1*, Toronto, CANADA, September, 1985.
- [30] Logicware Inc., *OS - Specific Details for MProlog on the Atari ST, PRE - Release BETA 2.1*, Toronto, CANADA, March, 1987.

- [31] Tomas Loranzo-Perez, "Automatic Planning of Manipulator Transfer Movements," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC 11, No. 10, October 1981, pp. 681-698.
- [32] M. Mazzon, A. Brighenti and W. Prendin, "TM 308 Towards Tetherless Inspection of Platforms," *Proceedings of ROV '85*, The Marine Technology Society, San Diego, Ca., April 2-4, 1985, pp. 222-227.
- [33] The Mobile Robot Laboratory Staff, "Towards Autonomous Vehicles," *The Robotics Institute 1985 Annual Research Review*, Robotics Institute, Carnegie-Mellon University, Pittsburg, Pa., 1986, pp. 33-49.
- [34] Andrew Moizer and Bernard Pagurek, "An Onboard Navigation System for Autonomous Underwater Vehicles," *Proceedings of the Intelligent Autonomous Systems Conference*, Amsterdam, The Netherlands, December 8-11, 1986.
- [35] Hans P. Moravec, "Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover," *Ph.D. Thesis, Stanford University*, September 1980, published as "Robot Rover Visual Navigation", *UMI Research Press*, Ann Arbor, Michigan, 1981.
- [36] Hans P. Moravec, "Locomotion, Vision, and Intelligence," *Robotics Research, The First Int'l Symposium*, MIT Press, Cambridge, Ma., 1984, pp. 215-224.
- [37] M. J. Morgan and T. L. Swanson, "Navigation Techniques Within Complex Offshore Structures," *Proceedings of the 11th. Annual Offshore Technology Conference*, Houston, Texas, April 30 - May 3, 1979, Vol II, pp. 1301-1306.
- [38] Nils J. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Co., Palo Alto, Ca., 1980.
- [39] John J. Nitao and Alexandre M. Parodi "An Intelligent Pilot for an Autonomous Vehicle System," *Proceedings of the Second Conference on Artificial Intelligence Applications*, IEEE Computer Society, Miami, December 11-13, 1985, pp. 176-183.
- [40] Nancy E. Orlando, "An Intelligent Robotics Control Scheme," *Proceedings of the 1984 American Control Conference*, San Diego, Ca., June 6-8, 1984, pp. 204-209.
- [41] Alexandre M. Parodi, "A Route Planning System for an Autonomous Vehicle," *Proceedings of the First International Conference on Artificial Intelligence Applications*, IEEE Computer Society, December 1984, pp. 51-56.

- [42] Robotics Systems International, Engineering Staff, *Private Consultation*, December 1984.
- [43] Ritchey Ruff and Narendra Ahuja, "Path Planning in a Three Dimensional Environment," *Seventh Int'l Conference on Pattern Recognition*, Montreal, Canada, August 1984, pp. 188-191.
- [44] G. T. Russell and J. Bugge, "Automatic Guidance of an Unmanned Submersible Using a Hierarchical Computer Control Strategy," *Oceans 81 Conference Record*, IEEE and Marine Technology Society, Boston, Ma., Sept 16-18, 1981, pp. 118-122.
- [45] J.E. Rygh and O. Kristiansen "Cleaning High Stress Areas of Complex Offshore Structures for Inspection Using Remote Operated Waterjetting Tools - Experiences From Sea Trials," *Norwegian Maritime Research*, No. 4, 1982, pp. 37-41.
- [46] J.E. Rygh and O. Kristiansen "Experiences with the Use of an ROV for Cleaning and Inspecting High Stress Areas of Complex Offshore Structures," *Oceans 82 Conference Record*, Marine Technology Society & IEEE Council on Oceanic Engineering, Washington DC, Sept. 20-22, 1982, pp. 73-77.
- [47] G. H. Sterling, A. O. P. Casbarian, N. L. Dodge and D. G. Godfrey, "Construction of the Cognac Platform, 1025 Feet of Water, Gulf of Mexico," *Proceedings of the 11th. Annual Offshore Technology Conference*, Houston, Texas, April 30 - May 3, 1979, Vol II, pp. 1169-1184.
- [48] Clinton Thomas, "Sea Trials and Operation of the ARCS Autonomous Submersible," *Proceedings of ROV '85*, The Marine Technology Society, San Diego, Ca., April 2-4, 1985, pp. 122-125.
- [49] Charles E. Thorpe and Larry H. Matthies, "Path Relaxation: Path Planning for a Mobile Robot," *Oceans 84 Conference Record*, Marine Technology Society, IEEE Ocean Engineering Society Washington DC, Sept. 10-12, 1984, pp. 576-581.
- [50] Robert Wernli, "The Silent World of the Undersea Robot," *1983 Decade of Robotics*, IFS (Publications) Ltd., Bedford, England, pp. 100-101.
- [51] A.S. Westneat, D.R. Blidberg and R.W. Corell, "Assessment of Technologies in Unmanned, Untethered Autonomous Submersibles," *ROV 83 Conference Record*, The Marine Technology Society, San Diego, Ca., March 14-17, 1983, pp. 267-271.

- [52] P. M. Winston, *Artificial Intelligence*, Addison-Wesley, Reading, Mass., 1984.
- [53] A. M. Wray, E. Ihnatowicz and D. R. Broome, "Development of a Manipulator for Automated Inspection of Subsea Structures," *ROV '84 Conference Record*, The Marine Technology Society, San Diego, Ca., May 14-18, 1984, pp. 37-41.

Appendix A

Prolog Source Code Listings

This appendix contains examples of the Prolog source code listings. There are seven modules that demonstrate the features of the navigation system. The modules are given in the following order:

astar5 The Prolog code for the path planning system. This code uses the facts in the network model and the description of the cells of free space to plan an optimal path between two points specified by the user.

model_builder This code takes a description of the platform geometry and generates facts of the form found in the module **scaled_demo_model**.

controller This module is used to coordinate the actions of the model builder and the path planner to dynamically generate the model and then use it for path planning.

demo_plat The description of the platform geometry and connectivity for the example case shown in figure 5.6.

demo_cells The facts that describe the cells of free space.

scaled_demo_model The network world model for the example structure depicted in the figure 5.6.

utilities Some commonly used utility predicates that are used by the other modules.

Description: These facts contain the partially developed paths used by the searching algorithm. The path is a list of nodes that the path has passed through. length is the length of the path traversed so far. Merit is the lower bound estimate of the total path length to the goal.

Predicate: temp_link(Node1:<integer>:R,
Node2:<integer>:R,
Scale:<real>:R).

Description: These facts are temporary links added to the world model that link dynamically created nodes to the rest of the world model. The Node arguments are labels of the nodes in the model. Scale represents the scaling factor to be applied to the link distance.

Predicate: temp_node(Label:<string>:R,
Position:<list>:R,
Face:<string>:R).

Description: These facts are temporary nodes added to the world model to describe arbitrary start and goal points for the planner. Label is an identifier used to describe the node. Position is a list containing the x,y,z coordinates of the node. Face is the label of the face of the platform that that the node is on.

Predicate: expand(Answer:<list>:RW,
Goal:<string>:R).

Description: This predicate advances the partial paths to the goal. The first clause checks the partial path facts to see if an appropriate solution has been found. If no acceptable solution is found then the second clause is used to advance the most promising partial path.

expand([Goal|Tail] Goal) :-
partial_path([Goal|Tail],_,_,Merit),
not better_solution(Merit).

expand(Answer,Goal) :-
select(Path,Length),
generate_alternatives(Path,Length,Goal),
expand(Answer,Goal).

Predicate: select(Path:<list>:W,
Length:<real>:W).

Description: This predicate finds the partial path with the lowest estimated path length, (ie. the partial path with the lowest merit value). The second predicate extracts the information from the chosen path and then deletes it from the list of candidates.

select(.,_) :-
set_up,
partial_path(.,_,_,Merit),
increment(index),
value(saveX,X),
Merit < X,
set_value(saveX,Merit),
value(index,I),
set_value(pos,I),
fail.

select(Path,Length) :-
increment(number_of_paths),
value(pos,Pos),
matching_statement((partial_path(Path,Length,_)),Pos,_),
del_nth_statement(partial_path/3,Pos),
!

Predicate: set_up.

Description: This predicate initializes the values for the select predicate.

set_up :-

```

set_value(index,0),
set_value(pos,1),
partial_path(...,Merit_init),
set_value(saveex,Merit_init),
!

```

```

*****

```

```

$ Predicate: generate_alternatives(Path:<list>:R,
Length:<real>:R,
Goal:<string>:R).

```

```

$ Description: This predicate is called to advance the partial
path that is passed to it. The predicate expands
the node at the head of the path and then adds
the appropriate partial_path facts to the
database. New partial paths are discarded if
they contain loops or reach a node that has already
been expanded with lower cost. The second
clause causes the predicate to succeed when all
new partial paths have been considered.

```

```

*****

```

```

generate_alternatives([Tip|Tail],Length,Goal) :-
planner_link(Tip,Next,Scale),
not member(Next,[Tip|Tail]),
link_length(Tip,Next,Ans),
New_length is Length + Ans * Scale,
check_path(Next,New_length),
link_length(Next,Goal,To_go),
value(scale_factor,Factor),
Merit is New_length + To_go * Factor,
add_statement(partial_path([Next,Tip|Tail],New_length,Merit),top),
fail.

```

```

generate_alternatives(...).

```

```

*****

```

```

$ Predicate: check_path(Node:<string>:R,
Length:<real>:R).

```

```

$ Description: This predicate checks the list of open nodes
to determine whether the input path reaches a node
that has already been opened. If the new path
has a longer length the predicate fails, otherwise
the list of open nodes is updated.

```

```

*****

```

```

check_path(Tip_node,Path_length) :-
open(Tip_node,Length),
Path_length > Length.

```

```

check_path(Tip_node,Path_length) :-
open(Tip_node,Length),
del_statement(open(Tip_node,Length)),
add_statement(open(Tip_node,Path_length),top).
+

```

```

check_path(Tip_node,Path_length) :-
add_statement(open(Tip_node,Path_length),top).
+

```

```

*****

```

```

$ Predicate: better_solution(Merit:<real>:R).

```

```

$ Description: This predicate checks the proposed solution
and succeeds if a partial path has a lower
merit figure than the input parameter.
(ie. failure means there is no possibility
of finding a better solution).

```

```

*****

```

```

better_solution(Merit) :-
partial_path(...,Other_merit),
Other_merit < Merit.

```

```

*****

```

```

$ Predicate: link_length(Node:<string>:R,
Node:<string>:R,
Path_length:<real>:M).

```

```

$ Description: This predicate calculates the actual straight
line distance between two nodes. An external
C routine is used for the Vax version for speed
and accuracy. The commented out code is used
for the Atari version where external predicates
cannot be used.

```

```

*****

```

```

link_length(N1,N2,Path_length) :-
planner_node(N1,[X1,Y1,Z1],_),
planner_node(N2,[X2,Y2,Z2],_),
compute_length(X1,Y1,Z1,X2,Y2,Z2,Path_length),
Temp is ((N1-X2)**2 + (Y1-Y2)**2 + (Z1-Z2)**2),
sq_root(Temp,Path_length).

```

```

*****

```

```

23 Predicate:
24 compute_length(X1,Y1,Z1:<real>:R,
25               X2,Y2,Z2:<real>:R,
26               length:<real>:W) .
27
28 Description:
29 This predicate computes the distace between
30 two points specified by their cartesian
31 coordinates in three space. An external C
32 function is used to increase performance.
33 The external routine must be removed for the
34 Atari version.
35
36 *****
37 *****
38 *****
39 *****
40 *****
41 *****
42 *****
43 *****
44 *****
45 *****
46 *****
47 *****
48 *****
49 *****
50 *****
51 *****
52 *****
53 *****
54 *****
55 *****
56 *****
57 *****
58 *****
59 *****
60 *****
61 *****
62 *****
63 *****
64 *****
65 *****
66 *****
67 *****
68 *****
69 *****
70 *****
71 *****
72 *****
73 *****
74 *****
75 *****
76 *****
77 *****
78 *****
79 *****
80 *****
81 *****
82 *****
83 *****
84 *****
85 *****
86 *****
87 *****
88 *****
89 *****
90 *****
91 *****
92 *****
93 *****
94 *****
95 *****
96 *****
97 *****
98 *****
99 *****
100 *****

```

```
external(compute_length/7,"cs/_compute_length(+f,+f,+f,+f,+f,+f,-f)
@//e/andrew/npool/c_files/compute_length.o -lm").
```

```
*****
Predicate:  planner_link(Node1:<string>:R,
                    Node2:<string>:W,
                    Scale:<real>:W)
*****
Description: This predicate takes all the links in the world
model and converts them to "two way" links
as required by the planner. This predicate also
joins the temporary links added by the planner
to the world model description.
*****
```

```

planner_link(Node1,Node2,Scale) :-
    mode_link(Node1,Node2,Scale);
    mode2_link(Node2,Node1,Scale);
    temp_link(Node1,Node2,Scale);
    temp_link(Node2,Node1,Scale).

```

```
Predicate: planner_node(Node_label:<string>:RW,  
                        Position:<list>:RW,  
                        Attributes:<list>:RW).
```

Description: This predicate takes the nodes stored in the static and dynamic models and combines them into one predicate (fact).

```
planner_node(Label,Position,Attributes) :-
    model_node(Label,Position,Attributes);
    temp_node(Label,Position,Attributes).
```

◆ ◆ ◆


```

%$*****
module model_builder.
%$
NAME:
%$
%$ model_builder
%$
%$*****
AUTHOR: Andrew Molzer
DATE: April 23, 1986
DESCRIPTION:
%$
This program constructs a series of facts that define
the world model of the oil platform structure from
facts that describe its the geometry. The facts that
are generated are in a form suitable for the path
planning/navigation program to use to plan safe
paths within the structure.
This program will generate the model_node, model_link
and face_plane facts required for the path planner.
These facts can be written to a file as Prolog
source code to form the world model.

```

```

all_symbolic.
global (no).

```

EXTERNALS:

```

export (make_model/0).
export (save_database/0).
export (model_link/3, model_node/3).
export (face_plane/3).
import (increment/1).
import (convex_poly/4, tube_number_2/3, joint/3).
import (face/2).
import (cpu_timer/2).
body.

```

LOCAL VARIABLES:

```

%$
dynamic (model_node/3)
dynamic (model_link/3)
dynamic (face_plane/3)
value (output_index)
value (node_index)
value (tolerance)
value (start)

```

```
value (stop).
```

PROPRIETARY NOTICE:

COPYRIGHT (c) 1987 Andrew D. Molzer, Ottawa, Canada.

The software contained herein, in all its forms, is protected by copyright. No portion, part or whole, may be reproduced, used or translated without expressed written permission of Andrew D. Molzer.

Andrew D. Molzer assumes no responsibility for the use or reliability of this software.

HISTORY:

```

date:      change:
----      -
86-6-30    Module changed to generate plane facts to determine
            a points position in the model.
86-8-27    Model generation timing added.
86-10-17   Code to write model to file added (from controller
            module).

```

```
/*object*/
```

```
Predicate: make_model().
```

Description: This predicate finds and writes all the nodes required for the planner. The first clause finds all the nodes at the centre of bracing triangles in the platform structure. All nodes that are found are added to the database as model_node facts. The second clause connects all the model_nodes that have been found using the platform and conceptual model information. All valid links are added to the database as model_link facts. The third clause generates the plane normal facts associated with each face plane description. These are added as face_plane facts. Timing information is given separately for all three phases of the model construction.

```
make_model :-
```

```

    make_init,
    nl, write("Starting to make the network model."), nl,
    cpu_timer(.,.),
    face(face_id, face_nodes),
    para_3(Three_joints, face_nodes),

```



```

%###
pick(Head, [Head|Tail], Tail).
pick(Head, [_|T], Tail) :-
    pick(Head, T, Tail).

```

```

%###
Predicate:      test_for_node(Joint:<list>:R,
                             Position:<list>:W).

```

```

Description:      This predicate tests whether there is a
                    valid world model node at the centre of the
                    triangle formed by joining the three joints
                    in the joint_list. If a valid node is found
                    (as dictated by the structure definition) then
                    its position is calculated and returned.
                    Otherwise the predicate fails.

```

```

%###
test_for_node([J1, J2, J3], Node_position) :-
    tube_member_2(J1, J2,
    tube_member_2(J1, J3,
    tube_member_2(J2, J3,
    get_joint_positions([J1, J2, J3], Position_list),
    valid_node(Position_list),
    get_node_pos(Position_list, Node_position).

```

```

%###
Predicate:      get_joint_positions(Label_1:<string>:R,
                             Label_2:<string>:R,
                             Label_3:<string>:R,
                             Position_list:<list>:W).

```

```

Description:      This predicate finds the position of three
                    joints in the platform model (defined by
                    their labels), and returns them as a list
                    (for easier use).

```

```

%###
get_joint_positions(J1, J2, J3, [Pos1, Pos2, Pos3]) :-
    joint(J1, Pos1,
    joint(J2, Pos2,
    joint(J3, Pos3,

```

```

%###
Predicate:      valid_node(Pos1:<list>:R,

```

```

%###
Pos2:<list>:R,
Pos3:<list>:R).

```

```

Description:      This predicate determines whether the
                    positions of the three joints describe
                    the corners of a valid model node.
                    The algorithm used is to construct two
                    vectors from the points and to check that
                    their cross product is not zero (or close
                    to it).

```

```

%###
valid_node([X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3], I) :-
    AX is X1-X2, AY is Y1-Y2, AZ is Z1-Z2,
    BX is X1-X3, BY is Y1-Y3, BZ is Z1-Z3,
    value(tolerance, Tolerance),
    (abs(AX*BZ - AZ*BY) > Tolerance;
    abs(AZ*BX - AX*BZ) > Tolerance;
    abs(AX*BY - AY*BX) > Tolerance),

```

```

%###
Predicate:      get_node_pos(Point1:<list>:R,
                             Point2:<list>:R,
                             Point3:<list>:R,
                             Node_position:<list>:W).

```

```

Description:      This predicate will find the position of the
                    node for the world model from the positions
                    of the three structure joints that make up
                    the corners of the triangle in which the model
                    node is located. The model node is located
                    at the centroid of the triangle described
                    by the input points.

```

```

%###
get_node_pos([X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3], [Node_X, Node_Y, Node_Z]) :-
    Node_X is (X1+X2+X3)/3,
    Node_Y is (Y1+Y2+Y3)/3,
    Node_Z is (Z1+Z2+Z3)/3.

```

```

%###
Predicate:      save_database().

```

```

Description:      This predicate saves the world model database
                    to a file specified by the user. The data is
                    stored as Microlog readable facts for easy
                    use by the planner (and other) Prolog programs.
                    The following assertions are written to the file:

```

of/de

Micro-D

Apr 12 23:28 1987 model_builder.pro Page 9

```

*****
**      Predicate:    goal      **
**      Description:  make the model and write it to a file if **
**                  desired.    **
*****
:-
make_model,
nl,write("Do you want to save the world model to a file?" ),nl,
read_token(Z),cut_input,
(prefix_of(Z,no):save_database),
write("End of model builder execution"),nl.

endmod /* model_builder */

```


module demo_plat

```

%#
NAME:
%#
%# demo_plat
%#
%#

```

AUTHOR: Andrew Molzer

DATE: October 16, 1986.

DESCRIPTION: This module contains the description of the joints and the connectivity for the demonstration model used in the thesis text. facts in this module are used by the model building program to construct the network world model.

all_global.

EXTERNALS:

```

export (joint/3).
export (tube_member_2/3).
export (face/2).
local (tube_member).
body.

```

PROPRIETARY NOTICE:

COPYRIGHT (c) 1987 Andrew D. Molzer, Ottawa, Canada.

The software contained herein, in all its forms, is protected by copyright. No portion, part or whole, may be reproduced, used or translated without expressed written permission of Andrew D. Molzer.

Andrew D. Molzer assumed no responsibility for the use or reliability of this software.

HISTORY:

```

date: change:
-----
86-10-16 original.
86-10-22 extra 'tower' added to model.

```

/*\$object*/

```

%#
%# Predicate: joint (Label:<string>:R,
%# Position:<list>:R,
%# Features:<list>:R)
%#

```

Description: These facts describe the location of the joints in the platform structure. The Position is a list of the joints cartesian coordinates.

% The following facts describe the joints at elevation 0 ft.

```

joint(a1,[0,0,0],[])
joint(a2,[10,0,0],[])
joint(a3,[20,0,0],[])
joint(a4,[0,10,0],[])
joint(a5,[10,10,0],[])
joint(a6,[20,10,0],[])
joint(a7,[30,0,0],[])
joint(a8,[30,10,0],[])

```

% The following facts describe the joints at elevation -10 ft.

```

joint(b1,[0,0,10],[])
joint(b2,[10,0,10],[])
joint(b3,[20,0,10],[])
joint(b4,[0,10,10],[])
joint(b5,[10,10,10],[])
joint(b6,[20,10,10],[])
joint(b7,[30,0,10],[])
joint(b8,[30,10,10],[])

```

% The following facts describe the joints at elevation -10 ft.

```

joint(c1,[0,0,20],[])
joint(c2,[10,0,20],[])
joint(c3,[20,0,20],[])
joint(c4,[0,10,20],[])
joint(c5,[10,10,20],[])
joint(c6,[20,10,20],[])
joint(c7,[30,0,20],[])
joint(c8,[30,10,20],[])

```

```

%#
%# Predicate: tube_member (Joint1:<string>:R,
%# Joint2:<string>:R,
%# Features:<list>:R)
%#

```

Description: These facts denote the connectivity of the platform joints. The Feature parameter allows additional information to be associated with the platform members if required.

```

%###*****
%

```

```

% The following facts represent the main pile members

```

```

tube_member(a1,b1,{}).
tube_member(b1,c1,{}).

tube_member(a2,b2,{}).
tube_member(b2,c2,{}).

tube_member(a3,b3,{}).
tube_member(b3,c3,{}).

tube_member(a4,b4,{}).
tube_member(b4,c4,{}).

tube_member(a5,b5,{}).
tube_member(b5,c5,{}).

tube_member(a6,b6,{}).
tube_member(b6,c6,{}).

tube_member(a7,b7,{}).
tube_member(b7,c7,{}).

tube_member(a8,b8,{}).
tube_member(b8,c8,{}).

```

```

% The following facts represent the framing for each 'layer'

```

```

% Layer 'a'

```

```

tube_member(a1,a2,{}).
tube_member(a2,a3,{}).
tube_member(a3,a7,{}).
tube_member(a4,a5,{}).
tube_member(a5,a6,{}).
tube_member(a6,a8,{}).
tube_member(a1,a4,{}).
tube_member(a2,a5,{}).
tube_member(a3,a6,{}).
tube_member(a7,a8,{}).
tube_member(a2,a4,{}).
tube_member(a3,a5,{}).
tube_member(a7,a6,{}).

```

```

% Layer 'b'

```

```

tube_member(b1,b2,{}).
tube_member(b2,b3,{}).
tube_member(b3,b7,{}).
tube_member(b4,b5,{}).
tube_member(b5,b6,{}).
tube_member(b6,b8,{}).
tube_member(b1,b4,{}).
tube_member(b2,b5,{}).

```

```

tube_member(b3,b6,{}).
tube_member(b7,b8,{}).
tube_member(b2,b4,{}).
tube_member(b3,b5,{}).
tube_member(b7,b6,{}).

```

```

% Layer 'c'

```

```

tube_member(c1,c2,{}).
tube_member(c2,c3,{}).
tube_member(c3,c7,{}).
tube_member(c4,c5,{}).
tube_member(c5,c6,{}).
tube_member(c6,c8,{}).
tube_member(c1,c4,{}).
tube_member(c2,c5,{}).
tube_member(c3,c6,{}).
tube_member(c7,c8,{}).
tube_member(c2,c4,{}).
tube_member(c3,c5,{}).
tube_member(c7,c6,{}).

```

```

% The following facts describe the diagonal members on the 'benis'.

```

```

tube_member(a1,b2,{}).
tube_member(a2,b3,{}).
tube_member(b1,c2,{}).
tube_member(b2,c3,{}).
tube_member(a3,b7,{}).
tube_member(b3,c7,{}).

```

```

tube_member(a4,b5,{}).
tube_member(a5,b6,{}).
tube_member(b4,c5,{}).
tube_member(b5,c6,{}).
tube_member(a6,b8,{}).
tube_member(b6,c8,{}).

```

```

% The following facts describe the diagonal members on the 'rows'.

```

```

tube_member(b1,a4,{}).
tube_member(c1,b4,{}).

```

```

tube_member(b2,a5,{}).
tube_member(c2,b5,{}).

```

```

tube_member(b3,a6,{}).
tube_member(c3,b6,{}).

```

```

tube_member(b7,a8,{}).
tube_member(c7,b8,{}).

```

```

%###*****

```

```

% Predicate: tube_member_2(Joint1 <string> BW,

```


Apr 12 23:28 1987 demo_plat.pro Page 5

```

% Joint2:<string>:RW,
% Feature:<list>:RW).
%
% Description: This predicate is used to make all the
%               'tube_member' facts two way.
%*****
%
% tube_member_2(N1,N2,Params) :-
%   tube_member(N1,N2,Params);
%   tube_member(N2,N1,Params).
%
%*****
%
% Predicate: face[Index:<string>:R,
%               Joint_list:<list>:R),
%
% Description: These facts indicate which joints in the
%               structure are associated with each face of
%               the platform structure. The face descriptions
%               are also used to define the convex polygonal
%               cells of free space.
%*****

```

```

% The 'layer' faces
% Layer at elevation 0 ft
face(layer_0_1,[a1,a2,a5,a4]).
face(layer_0_2,[a2,a3,a6,a5]).
face(layer_0_3,[a3,a7,a8,a6]).
%
% Layer at elevation -10 ft
face(layer_10_1,[b1,b2,b5,b4]).
face(layer_10_2,[b2,b3,b6,b5]).
face(layer_10_3,[b3,b7,b8,b6]).
%
% Layer at elevation -20 ft
face(layer_20_1,[c1,c2,c5,c4]).
face(layer_20_2,[c2,c3,c6,c5]).
face(layer_20_3,[c3,c7,c8,c6]).
%
% The 'bent' faces
% The first bent plane 'A'
face(bent_a1,[a1,b1,b2,a2]).
face(bent_a2,[a2,a3,b3,b2]).
face(bent_a3,[b1,c1,c2,b2]).
face(bent_a4,[b2,b3,c3,c2]).
face(bent_a5,[a3,b7,b7,a3]).
face(bent_a6,[b3,b7,c7,c3]).

```

Apr 12 23:28 1987 demo_plat.pro Page 6

```

% The second bent plane 'B'
face(bent_b1,[a4,a5,b5,b4]).
face(bent_b2,[a5,a6,b6,b5]).
face(bent_b3,[b4,b5,c5,c4]).
face(bent_b4,[b5,b6,c6,c5]).
face(bent_b5,[a6,a8,b8,b6]).
face(bent_b6,[b6,b8,c8,c6]).
%
% The row faces
% Row 1
face(row_1_1,[a1,a4,b4,b1]).
face(row_1_2,[b1,b4,c4,c1]).
%
% Row 2
face(row_2_1,[a2,a5,b5,b2]).
face(row_2_2,[b2,b5,c5,c2]).
%
% Row 3
face(row_3_1,[a3,a6,b6,b3]).
face(row_3_2,[b3,b6,c6,c3]).
%
% Row 4
face(row_4_1,[a7,a8,b8,b7]).
face(row_4_2,[b7,b8,c8,c7]).
endmod /* demo_plat */

```

```
module demo_cells.
```

```
NAME:
*****
* demo_cells *
*****
```

```
AUTHOR: Andrew Molzer
```

```
DATE: October 16, 1986.
```

```
DESCRIPTION: This module contains the description of the convex
cells that define the conceptual model of the
demonstration platform used in the thesis.
Different scaling factors have been allocated to the
different cells.
```

```
all_global.
```

```
*****
```

```
export (convex_poly/4).
```

```
body.
```

```
*****
```

```
LOCAL VARIABLES:
```

```
PROPRIETARY NOTICE:
```

```
COPYRIGHT (c) 1987 Andrew D. Molzer, Ottawa, Canada.

The software contained herein, in all its forms, is protected
by copyright. No portion, part or whole, may be reproduced,
used or translated without expressed written permission of
Andrew D. Molzer.
```

```
Andrew D. Molzer assumes no responsibility for the use or
reliability of this software.
```

```
HISTORY:
```

```
date: change:
----
86-10-16 original.
86-10-17 different scaling factors added to cells in new
module name.
86-10-22 extra cells added.
```

```
*****
```

```
Predicate: convex_poly(label:<string>:R,
Interior_point:<list>:R,
Face_list:<list>:R,
Cell_scale_factor:<real>:R)
```

```
Description: These facts describe the three dimensional
convex polyhedral cells that make up the platform
structure. These capture the high level
'conceptual' model.
The interior point is any arbitrary point within
the cell that is used in the generation of the
face plane normals.
The face list is a list of the face (planes) that
describe the surfaces of the cells.
The cell scale factor is used as a baseline figure
when assigning link scaling factors to the links
that pass through the cell.
```

```
*****
```

```
convex_poly(cell_1,[5,5,5],
[layer_0_1,layer_10_1,bent_a1,bent_b1,row_1_1,row_2_1],1),
convex_poly(cell_2,[15,5,5],
[layer_0_2,layer_10_2,bent_a2,bent_b2,row_2_1,row_3_1],2.5e0),
convex_poly(cell_3,[25,5,5],
[layer_0_3,layer_10_3,bent_a3,bent_b3,row_3_1,row_4_1],1),
convex_poly(cell_4,[5,5,15],
[layer_10_1,layer_20_1,bent_a3,bent_b3,row_1_2,row_2_2],1),
convex_poly(cell_5,[15,5,15],
[layer_10_2,layer_20_2,bent_a4,bent_b4,row_2_2,row_3_2],1),
convex_poly(cell_6,[25,5,15],
[layer_10_3,layer_20_3,bent_a6,bent_b6,row_3_2,row_4_2],1).
```

```
endmod /* demo_cells */
```


Description: These facts describe the links in the network model. The nodes are represented by the node labels defined in the model_node facts. The scale_factor is used to calculate the effective length of the path segment.
These facts are generated automatically from the platform description by the model building program.

```

node_link(1,7,1).
node_link(1,8,1).
node_link(1,19,1).
node_link(1,20,1).
node_link(1,29,1).
node_link(1,30,1).
node_link(1,41,1).
node_link(1,42,1).
node_link(1,45,1).
node_link(1,46,1).
node_link(2,7,1).
node_link(2,8,1).
node_link(2,19,1).
node_link(2,20,1).
node_link(2,29,1).
node_link(2,30,1).
node_link(2,41,1).
node_link(2,42,1).
node_link(2,45,1).
node_link(2,46,1).
node_link(7,19,1).
node_link(7,20,1).
node_link(7,29,1).
node_link(7,30,1).
node_link(7,41,1).
node_link(7,42,1).
node_link(7,45,1).
node_link(7,46,1).
node_link(8,19,1).
node_link(8,20,1).
node_link(8,29,1).
node_link(8,30,1).
node_link(8,41,1).
node_link(8,42,1).
node_link(8,45,1).
node_link(8,46,1).
node_link(19,29,1).
node_link(19,30,1).
node_link(19,41,1).
node_link(19,42,1).
node_link(19,45,1).
node_link(19,46,1).
node_link(20,29,1).
node_link(20,30,1).
node_link(20,41,1).

```

```

node_link(20,42,1).
node_link(20,45,1).
node_link(20,46,1).
node_link(29,41,1).
node_link(29,42,1).
node_link(29,45,1).
node_link(29,46,1).
node_link(30,41,1).
node_link(30,42,1).
node_link(30,45,1).
node_link(30,46,1).
node_link(41,45,1).
node_link(41,46,1).
node_link(42,45,1).
node_link(42,46,1).
node_link(3,9,2,5E0).
node_link(3,10,2,5E0).
node_link(3,21,2,5E0).
node_link(3,22,2,5E0).
node_link(3,31,2,5E0).
node_link(3,32,2,5E0).
node_link(3,45,2,5E0).
node_link(3,46,2,5E0).
node_link(3,49,2,5E0).
node_link(3,50,2,5E0).
node_link(4,9,2,5E0).
node_link(4,10,2,5E0).
node_link(4,21,2,5E0).
node_link(4,22,2,5E0).
node_link(4,31,2,5E0).
node_link(4,32,2,5E0).
node_link(4,45,2,5E0).
node_link(4,46,2,5E0).
node_link(4,49,2,5E0).
node_link(4,50,2,5E0).
node_link(9,21,2,5E0).
node_link(9,22,2,5E0).
node_link(9,31,2,5E0).
node_link(9,32,2,5E0).
node_link(9,45,2,5E0).
node_link(9,46,2,5E0).
node_link(9,49,2,5E0).
node_link(9,50,2,5E0).
node_link(10,21,2,5E0).
node_link(10,22,2,5E0).
node_link(10,31,2,5E0).
node_link(10,32,2,5E0).
node_link(10,45,2,5E0).
node_link(10,46,2,5E0).
node_link(10,49,2,5E0).
node_link(10,50,2,5E0).
node_link(21,31,2,5E0).
node_link(21,32,2,5E0).
node_link(21,45,2,5E0).
node_link(21,46,2,5E0).
node_link(21,49,2,5E0).

```

```

model_link(21,50,2,520)
model_link(22,31,2,520)
model_link(22,32,2,520)
model_link(22,45,2,520)
model_link(22,46,2,520)
model_link(22,49,2,520)
model_link(22,50,2,520)
model_link(31,45,2,520)
model_link(31,46,2,520)
model_link(31,49,2,520)
model_link(31,50,2,520)
model_link(32,45,2,520)
model_link(32,46,2,520)
model_link(32,49,2,520)
model_link(32,50,2,520)
model_link(45,49,2,520)
model_link(45,50,2,520)
model_link(46,49,2,520)
model_link(46,50,2,520)
model_link(5,11,1)
model_link(5,12,1)

```

```

% The intervening facts are removed to keep the file size reasonable
% for this thesis presentation

```

```

model_link(39,55,1)
model_link(39,56,1)
model_link(40,51,1)
model_link(40,52,1)
model_link(40,55,1)
model_link(40,56,1)
model_link(51,55,1)
model_link(51,56,1)
model_link(52,55,1)
model_link(52,56,1)

```

```

*****
Predicate: face_plane(Label:<string>:R,
Normal_vector:<list>:R,
Plane_offset:<real>:R).

```

```

Description: These facts geometrically define the planes
that the nodes of the models lie in. These
planes represent the surfaces of the model cells.
The normal vector is represented by the vector
components.
These facts are generated automatically from
the platform description by the model building
program.

```

```

*****
face_plane(layer_0_1,[0,0,100],0)
face_plane(layer_0_2,[0,0,100],0)

```

```

face_plane(layer_0_3,[0,0,100],0)
face_plane(layer_10_1,[0,0,100],1000)
face_plane(layer_10_2,[0,0,100],1000)
face_plane(layer_10_3,[0,0,100],1000)
face_plane(layer_20_1,[0,0,100],2000)
face_plane(layer_20_2,[0,0,100],2000)
face_plane(layer_20_3,[0,0,100],2000)
face_plane(bent_a1,[0,100,0],0)
face_plane(bent_a2,[0,-100,0],0)
face_plane(bent_a3,[0,100,0],0)
face_plane(bent_a4,[0,-100,0],0)
face_plane(bent_a6,[0,-100,0],0)
face_plane(bent_b1,[0,-100,0],1000)
face_plane(bent_b2,[0,-100,0],1000)
face_plane(bent_b3,[0,-100,0],1000)
face_plane(bent_b4,[0,-100,0],1000)
face_plane(bent_b5,[0,-100,0],1000)
face_plane(bent_b6,[0,-100,0],1000)
face_plane(row_1_1,[100,0,0],0)
face_plane(row_1_2,[100,0,0],0)
face_plane(row_2_1,[100,0,0],1000)
face_plane(row_2_2,[100,0,0],1000)
face_plane(row_3_1,[100,0,0],2000)
face_plane(row_3_2,[100,0,0],2000)
face_plane(row_4_1,[100,0,0],3000)
face_plane(row_4_2,[100,0,0],3000)

```

```

endmod /* scaled_demo_model */

```



```

*****
Predicate:      print_delta_cpu(Message:<string>:R,
                          Start,Stop:<string>:R).
Description:    This predicate takes as inputs an output
                message string and the names
                of two value variables that are assumed to
                contain marked cpu times. The difference
                between them is calculated and printed.
*****
print_delta_cpu(Message,Start_value,Stop_value) :-
    value(Start_value,Start_time),
    value(Stop_value,Stop_time),
    Delta is Stop_time - Start_time,
    Secs is Delta div 1000,
    Frac is Delta mod 1000,
    nl,write(Message),write(Secs),write("."),write(Frac),
    write(" secs."),nl,
    !.
*****

```

```

*****
Predicate:      more(Prompt:<string>:R).
Description:    Predicate to determine whether the user
                wants to try another loop or solution to
                the user is prompted by the input string
                and then any prefix of "no" will cause the
                predicate to succeed, otherwise it fails.
*****
more(Prompt_string) :-
    nl,
    write(Prompt_string),nl,
    read_token(X),
    cut_input,
    nl,
    prefix_of(X,no).
*****

```

```

*****
Predicate:      number_write(Number:<real>:R,
                          Places:<Integer>:R)
Description:    This predicate takes a number in the floating
                point form and writes it as a decimal number.
                The number of decimal places for the output
                is indicated by the second argument.
*****
number_write(Number,Places) :-

```

```

Mult is 10 ** Places,
Temp is round(Number * Mult),
Int is Temp div Mult,
Frac is Temp mod Mult,
write(Int),write("."),write(Frac),
!.

```

```

*****
Predicate:      cpu_timer(TOTAL_TIME:<Integer>:W,
                          ELAPSED_TIME:<Integer>:W)
Description:    This predicate will retrieve the total
                cpu time used by the process and the elapsed time since
                the last call was made; time is in seconds.
                Thus to time a piece of code do:
                cpu_timer(...).
                ....
                <your code>,
                ....
                cpu_timer(...,USED),
                courtesy of Gerald Karam.
                Note: must be commented out for Atari port.
*****
external(cpu_timer/2,"cs/_cpu_timer(-n,-n)/e/andrew/wprol/c_files/cpu_timer.o").

```

```

*****
Predicate:      reverse(L1:<list>:R,
                          L2:<list>:W).
Description:    Reverse the elements of the first list into
                the second list.
*****

```

```

reverse(L1,L2) :-
    fast_reverse(L1,[],L2).
fast_reverse([X|L],L2,L3) :-
    fast_reverse(L,[X|L2],L3).
fast_reverse([],L,L).

```

```

*****
Predicate:      sq_root(Number:<real>:R,
                          Root:<real>:W).
*****

```


78 Description: This predicate calculates the square root of a
79 number using five iterations of the Newton-
80 Raphson method. This gives acceptable accuracy
81 for numbers whose square root is less than 150.
82 Required for the Atari port.

78#####

sq_root(Number,Root).
X1 is 15*NUMBER/60.
X2 is X1/2+Number/(2*X1).
X3 is X2/2+Number/(2*X2).
X4 is X3/2+Number/(2*X3).
Root is X4/2+Number/(2*X4).

endmod /* utilities */

Appendix B

Platform Drawings

This appendix contains example drawings for a typical eight leg offshore platform. These drawings were obtained from the Brown & Root Company [5] who retain all rights.

KEY PLAN

2025-2026

~~CLASSIFICATION ROW. 2.~~

[illegible]

...the ... of ...

END

06-07-89

FIN