

## INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

# UMI

A Bell & Howell Information Company  
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA  
313/761-4700 800/521-0600



**SPOTT:**  
**A REAL-TIME, DISTRIBUTED AND SCALABLE ARCHITECTURE**  
**FOR AUTONOMOUS MOBILE ROBOT CONTROL**

**John S. Zelek**

Centre for Intelligent Machines  
Department of Electrical Engineering  
McGill University

1996

A Thesis submitted to the Faculty of Graduate Studies and Research  
in partial fulfillment of the requirements of the degree of  
Doctor of Philosophy

© JOHN ZELEK, 1996



National Library  
of Canada

Acquisitions and  
Bibliographic Services

395 Wellington Street  
Ottawa ON K1A 0N4  
Canada

Bibliothèque nationale  
du Canada

Acquisitions et  
services bibliographiques

395, rue Wellington  
Ottawa ON K1A 0N4  
Canada

*Your file Votre référence*

*Our file Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-30432-9



## Abstract

---

A mobile robot control architecture called SPOTT<sup>1</sup> is proposed and implemented as a real-time system of concurrently executing and co-operating modules. What distinguishes SPOTT from other behavioral architectures is that it is able to guarantee task completion for navigational tasks under many different scenarios. SPOTT provides a bridge for linking behavioral (i.e., reactive) and symbolic control and has actually been interfaced with the logical reasoning system called COCOLOG. One of the roles of the symbolic reasoner is to help guarantee task completion in the situations where SPOTT is not able to solely do so. In essence, SPOTT is a real-time AI system which is responsible for dynamically adapting to changing environmental circumstances in order to successfully execute and complete a set of navigational tasks for an autonomous mobile robot.

SPOTT consists of a behavioral controller, a local dynamic path planner, and a global path planner, as well as a map database and a graphical user interface. The behavioral control formalism is called TR+ and is based on an adaptation and extension of the Teleo-Reactive (TR) formalism. TR+ rules make decisions which affect actuator control and map database maintenance. A dynamic local path planner continually polls the map database in order to navigate around newly encountered obstacles. The local dynamic path planner is based on a potential field method using harmonic functions, which are guaranteed to have no spurious local minima. The global planning module advises the local planning module in order to position and project the global goal onto the local border. A real-time and parallel implementation of SPOTT using a message passing software package called PVM has been developed and tested across a collection of ten to fifteen heterogeneous workstations. Navigational experiments have consisted of moving the robot in an office and laboratory environment to known spatial locations with no or a partial a priori map.

---

<sup>1</sup>A System which integrates Potential fields for planning On-line with TR+ program control in order to successfully execute a general suite of Task commands.

## Résumé

---

Une architecture de commande de robot mobile nommée SPOTT est proposée dans ce manuscrit. La réalisation de cette architecture consiste en un système temps-réel composé de modules coopérant et s'exécutant en parallèle. SPOTT peut garantir l'achèvement d'une tâche de navigation donnée dans des situations très variées; ceci le distingue d'autres architectures réactives. SPOTT permet également d'utiliser conjointement des techniques de contrôle réactif et symbolique, et est actuellement interfacé avec COCOLOG utilisé ici comme moteur d'inférences logiques. Ce moteur d'inférence garantit l'achèvement des tâches lorsque SPOTT ne peut le faire de lui-même. Essentiellement, SPOTT est un système d'IA qui peut s'adapter dynamiquement aux changements de l'environnement pour compléter les tâches de navigation nécessaires à un robot mobile.

SPOTT contient un système de commande réactif, un planificateur de trajectoires locales dynamiques, un planificateur de trajectoires globales, ainsi qu'une base de données cartographiques et une interface graphique. Le formalisme réactif TR+ provient d'une adaptation et d'une extension du formalisme Téléo-Réactif (TR). Les règles TR+ prennent des décisions qui agissent sur les contrôleurs du robot et le maintien de la base de données cartographiques. Le planificateur de trajectoires locales vérifie continuellement la base de données cartographiques pour modifier la trajectoire du robot lorsque de nouveaux obstacles sont découverts. Ce planificateur utilise une technique se servant de champs de potentiels et de fonctions harmoniques qui garantie une solution sans minimums locaux. Le planificateur de trajectoires globales projète le but global de la trajectoire sur la bordure locale et avise le planificateur local. Une implémentation parallèle temps-réel de SPOTT, utilisant le logiciel de communication inter-procédés PVM, a été élaborée. Un réseau hétérogène de 10 à 15 ordinateurs est utilisé pour l'expérimentation. Les expériences de navigation consistent à déplacer le robot dans un environnement de laboratoire ou de bureau à un point donné, et ceci avec ou sans données cartographiques a priori.

## Dedication

---

This dissertation is dedicated to

*my wife*

**Laura**

*for her support along the way,*

and

*my parents*

**Joseph and Maria Zelek**

*for encouraging me to pursue my doctorate.*

## Acknowledgements

---

*My thanks are due to*

**Professor Martin Levine**

*for his insights and guidance, as well as providing me with  
the opportunity to explore this avenue of research:*

*to*

**Professor Gregory Dudek,**

**Mike Kelly, and Kenong Wu**

*for their inspiration through numerous philosophical discussions;*

*to*

**Don Bui, and Paul MacKenzie**

*for the use of and consultation about their sensor algorithm implementations;*

*to*

**Marc Bolduc, Thierry Baron,**

**Nicholas Roy, and Michael Daum**

*for their technical assistance at various stages;*

*and to*

**all the people I have crossed paths with**

**during my stay at the**

**Centre for Intelligent Machines.**

I would also like to express my gratitude to the various funding agencies that have supported my work over the years. Specifically, to NSERC for their assistance through a graduate scholarship. This research was partially supported by the *Natural Sciences and Engineering Research Council* and by the *National Centres of Excellence Program* through IRIS (*Institute for Robotics and Intelligent Systems*). In particular, this research is part of the *Dynamic Reasoning, Navigation and Sensing for Mobile Robots* project under the ISDE (*Integrated Systems in Dynamic Environments*) theme.

## TABLE OF CONTENTS

---

Abstract . . . . .	ii
Résumé . . . . .	iii
Dedication . . . . .	iv
Acknowledgements . . . . .	v
LIST OF FIGURES . . . . .	x
LIST OF TABLES . . . . .	xiv
CHAPTER 1. Introduction . . . . .	1
1. Types of Navigational Tasks . . . . .	6
2. The SPOTT Robot Control System . . . . .	8
3. Claims of Originality . . . . .	12
4. Reader's Guide . . . . .	14
CHAPTER 2. Background . . . . .	15
1. Review of Behavioral Robot Architectures . . . . .	16
2. Interfacing to a Robot Control Architecture . . . . .	23
2.1. Role of the Operator . . . . .	23
2.2. The Complexity of the Environment . . . . .	24
2.3. The Mobile Robot . . . . .	25
2.3.1. Vision . . . . .	27
2.3.2. Acoustic (Sonar) . . . . .	27
2.3.3. Laser Rangefinders . . . . .	28
2.3.4. Other Types of Sensors . . . . .	29
3. Robot Control Issues . . . . .	31
3.1. Mobile Robot Navigational Problems . . . . .	32
	vi

3.1.1.	Path Planning . . . . .	33
3.1.2.	Path Execution . . . . .	33
3.1.3.	Robot Position Estimation . . . . .	34
3.1.4.	Map Building . . . . .	35
3.2.	Real-time AI . . . . .	36
4.	The SPOTT Robot Control System . . . . .	38
CHAPTER 3. Control: Teleo-Reactive+ Programs . . . . .		40
1.	Teleo-Reactive Programs . . . . .	41
1.1.	TR Program Syntax . . . . .	43
1.2.	Graph Representation . . . . .	44
1.3.	TR Program Interpretation and Execution . . . . .	46
1.4.	Circuit Semantics . . . . .	48
1.5.	TR Program Examples . . . . .	50
1.6.	Advantages and Disadvantages of the TR formalism . . . . .	53
1.6.1.	Advantages . . . . .	53
1.6.2.	Disadvantages . . . . .	53
2.	Teleo-Reactive+ Programs . . . . .	55
2.1.	TR+ Syntax . . . . .	61
2.2.	TR+ Programs for Robot Control . . . . .	62
CHAPTER 4. Path Planning . . . . .		74
1.	Path Planning Approaches . . . . .	78
2.	Dynamic Path Planning Problem . . . . .	80
3.	Path Planning Using Potential Fields . . . . .	82
3.1.	Biological Inspiration . . . . .	82
3.2.	Formulating a Potential Function . . . . .	82
3.3.	Summation of Potentials Approach . . . . .	83
4.	Harmonic Functions as Potential Functions . . . . .	87
4.1.	Computing Harmonic Functions . . . . .	87
4.1.1.	Number of Iterations Reduced by Methods-of-Relaxation . . . . .	90
4.1.2.	A Good Initial Guess . . . . .	95
5.	Trajectory Generation Using Harmonic Functions . . . . .	98
6.	Guaranteeing Proper Control . . . . .	101
7.	Why is the Potential Field a Local Path Planner? . . . . .	104
		vii

8.	Global Path Planning . . . . .	106
9.	Local and Global Path Planner Interaction . . . . .	110
9.1.	Four Typical Scenarios . . . . .	110
9.2.	Reachability . . . . .	116
9.3.	Algorithm: Projection of Global Goal onto Local Border . . . . .	117
10.	Future Research Issues . . . . .	122
CHAPTER 5. A Task Command Language Lexicon . . . . .		123
1.	Verbs . . . . .	126
1.1.	Motion Verbs . . . . .	126
1.2.	Vision Verbs . . . . .	126
2.	Spatial Locations and Object Descriptions . . . . .	128
3.	Prepositions . . . . .	131
3.1.	Quantifying Prepositional Expressions . . . . .	134
3.1.1.	Directional and Distal Prepositions . . . . .	136
3.1.2.	Trajectory Prepositions . . . . .	140
4.	The Task Command . . . . .	143
CHAPTER 6. Implementation . . . . .		145
1.	Parallelism with PVM . . . . .	147
2.	The SPOTT System . . . . .	151
2.1.	Graphical User Interface . . . . .	154
2.1.1.	Drawing Graphs with Dotty . . . . .	157
2.2.	Using the GUI as SPOTT's Main Controller . . . . .	158
2.3.	Local Path Planning . . . . .	163
2.4.	Map Database . . . . .	165
2.5.	The TR+ Interpreter . . . . .	169
2.5.1.	Programming TR+ Programs . . . . .	170
2.6.	RoboDaemon . . . . .	171
3.	Reasoning System Interface . . . . .	173
4.	Modularity and Portability . . . . .	175
CHAPTER 7. Experiments . . . . .		176
1.	Message-Passing Costs on a Network . . . . .	178
2.	Local Path Planning . . . . .	182
3.	Perception . . . . .	187
		viii

▼	3.1. Mapping Using Sonar . . . . .	187
▲	3.2. Localization Using Sonar . . . . .	190
	3.3. Mapping Using QUADRIS . . . . .	191
	4. Navigational Experiments . . . . .	194
	5. Off Site Experiment . . . . .	200
	6. Lessons Learned . . . . .	205
	CHAPTER 8. Conclusions . . . . .	207
	1. Contributions . . . . .	209
	2. Robot Control Issues . . . . .	211
	3. Future Research Possibilities . . . . .	213
	4. Summary . . . . .	216
	REFERENCES . . . . .	217
	APPENDIX A. Harmonic Function Computability . . . . .	225
	1. Neumann Boundary . . . . .	226
	2. Dirichlet Boundary . . . . .	227
▼	APPENDIX B. TR+ Conditions and Actions Implemented as Part of SPOTT . .	229
▲	1. Conditions . . . . .	229
	2. Actions . . . . .	232
	3. Variables . . . . .	233



## LIST OF FIGURES

---

1.1	The SPOTT System . . . . .	4
1.2	Types of Navigational Tasks . . . . .	6
1.3	What SPOTT Can and Cannot Do . . . . .	7
1.4	System Overview . . . . .	9
2.1	Classical Robot Control Architecture . . . . .	15
2.2	Brooks' Subsumption Architecture . . . . .	17
2.3	Brooks' Proposed Hierarchy of Behaviors . . . . .	18
2.4	Traditional vs. Behavioral Robot Control Architecture . . . . .	19
2.5	Behavioral Architecture Example . . . . .	20
2.6	Three-Layered Hierarchical Architecture . . . . .	21
2.7	Interfaces to a Robot Control Architecture . . . . .	23
2.8	Environmental Complexity . . . . .	25
2.9	Mobile Robot Sensors. . . . .	26
3.1	K-B Model of an Embedded Agent . . . . .	43
3.2	A TR Tree . . . . .	45
3.3	A TR Program . . . . .	46
3.4	A TR Sequence Created From a TR Tree . . . . .	47
3.5	The Main Program and a Subroutine Program . . . . .	48
3.6	Implementing a TR Sequence as a Circuit . . . . .	49
3.7	A TR Tree for Controlling a Thermostat . . . . .	50
3.8	A Simple TR Tree for Controlling a Robot . . . . .	51
3.9	Circuit Semantics for a Simple Robot Controller . . . . .	52

3.10	A TR+ Parallel Action and the Associated Circuit Semantics . . . . .	57
3.11	TR extended vs. TR+ Trees for ANDing Conditions . . . . .	60
3.12	A Simple TR+ Program for Navigation to a Specified Location . . . . .	63
3.13	A Typical TR+ Program for Mobile Robot Navigation . . . . .	66
3.14	The Main Routine for a More Complicated TR+ Program . . . . .	67
3.15	TR+ Subroutines Responsible for Maintaining the Potential Field . . . . .	67
3.16	TR+ Subroutines for Mapping . . . . .	68
3.17	TR+ Subroutines for Setting the Goals for Navigation . . . . .	69
3.18	TR+ Subroutines for Performing Reactivity . . . . .	70
3.19	TR+ Subroutines for Deciding Search Strategies . . . . .	71
3.20	TR+ Subroutine for Performing Room Search . . . . .	72
3.21	TR+ Subroutine for Performing Intelligent Teleo-Operation . . . . .	73
3.22	TR+ Subroutine for Performing Random Search . . . . .	73
4.1	Path Planning Within SPOTT . . . . .	75
4.2	Dynamic Path Planning Problem . . . . .	81
4.3	Summation of Potentials: Close Objects . . . . .	84
4.4	Maximum Summation of Potentials: Close Objects . . . . .	85
4.5	Potential Field Using Neumann boundary conditions. . . . .	90
4.6	Potential Field Using Dirichlet boundary conditions. . . . .	91
4.7	Iteration Kernel and Resistive Grid . . . . .	92
4.8	Types of Changes to the Obstacle and Goal Configuration . . . . .	97
4.9	Quick Calculation of Trajectory . . . . .	99
4.10	Multi-Resolution Potential Fields . . . . .	103
4.11	Sliding Local Path Planner . . . . .	105
4.12	CAD Map of CIM . . . . .	106
4.13	Abstract Graph of CAD Map . . . . .	107
4.14	Calculating Distances for Global Path Planning . . . . .	109
4.15	Local and Global Path Planner Interaction: First Typical Case . . . . .	112
4.16	Local and Global Path Planner Interaction: Second Typical Case . . . . .	113

4.17	Local and Global Path Planner Interaction: Third Typical Case . . .	114
4.18	Local and Global Path Planner Interaction: Fourth Typical Case . .	115
4.19	Reachability and Blocking Constraints . . . . .	118
4.20	Goal Projection: Goal in Same Node but not in the Potential Field .	120
4.21	Goal Projection: Locally Unreachable Goal . . . . .	120
4.22	Goal Projection: General Case . . . . .	121
5.1	2D Models for Representing the Spatial Area Defining the Destination	128
5.2	Geons . . . . .	129
5.3	Spatial Expression Regions Defined as Goals for Path Planning . . .	137
5.4	Directional Preposition Definitions . . . . .	138
5.5	Distance Preposition Definitions . . . . .	139
5.6	Trajectory Preposition Bias on Steepest Gradient Descent . . . . .	141
5.7	Trajectory Preposition Definitions . . . . .	142
5.8	The Task Command Lexicon . . . . .	144
6.1	A Hypothetical "Parallel Virtual Machine" . . . . .	149
6.2	Implementation Modules . . . . .	153
6.3	SPOTT's Graphical User Interface . . . . .	156
6.4	SPOTT's GUI State Transition Diagram . . . . .	162
6.5	Potential Field Master-Slave Configuration . . . . .	164
6.6	SPOTT and the Implementation of the Map Database . . . . .	168
6.7	TR+ Programming With TR+edit . . . . .	171
6.8	Integration of COCOLOG with SPOTT . . . . .	174
7.1	PVM Bandwidth Fluctuation Tests . . . . .	180
7.2	Potential Field Computation Times Versus Grid Size . . . . .	184
7.3	Computation Times for a Single Iteration . . . . .	185
7.4	Relation of Initial Guess to Convergence Rate . . . . .	186
7.5	Sonar Mapping in Open Spaces . . . . .	188
7.6	Sonar Mapping in Narrow Hallways . . . . .	189

7.7	Good Locations to Perform Localization . . . . .	190
7.8	The QUADRIS System . . . . .	191
7.9	QUADRIS Range Data . . . . .	192
7.10	Dynamic Mapping and Trajectory Determination . . . . .	195
7.11	Autonomous Navigation with No A Priori Map . . . . .	196
7.12	Autonomous Navigation with a Partial A Priori Map . . . . .	197
7.13	Projection Onto a Point or a Line . . . . .	199
7.14	Map of the Demonstration Environment . . . . .	201
7.15	Example of an Execution During the IRIS-PRECARN Demonstration	202
7.16	Sonar Reflections . . . . .	203
A.1	Potential Function Extent . . . . .	226

## LIST OF TABLES

---

3.1	Truth Table for the AND Binary Logical Operators . . . . .	58
3.2	Truth Table for the OR Binary Logical Operators . . . . .	59
3.3	Truth Table for the Unary Logical Operators . . . . .	59
5.1	The Minimal Spanning Set of English Prepositions . . . . .	132
7.1	PVM Bandwidth Tests At Peak and Low Periods . . . . .	179

## CHAPTER 1

---

### Introduction

There is a variety of potential applications for mobile robots in such diverse areas as forestry, space, nuclear reactors, environmental disasters, industry, and offices. Tasks in these environments are hazardous to humans, remotely located, or tedious. Potential tasks for autonomous mobile robots include maintenance, delivery, and security surveillance, which all require some form of intelligent navigational capabilities. A mobile robot will be a useful addition to these domains only when it is capable of functioning robustly under a wide variety of environmental conditions and able to operate without human intervention for long periods of time. The environments in which mobile robots must function are dynamic, unpredictable and not completely specifiable by a map beforehand. In order for a mobile robot to successfully complete a set of tasks, it must dynamically interact with the environment and adapt to changing circumstances.

Specifically, this thesis is concerned with a robot control architecture for autonomous mobile robot navigation in an office and laboratory environment. The only prior knowledge that the robot may have of its environment is a map of the permanent unchanging structures (i.e., walls), as would be presented by an architectural CAD map. The robot control architecture needs to make timely decisions based on the sensed data in order to continually plan and execute robot motions in order to complete the requested task. Biological creatures apparently execute many tasks in the world by using a combination of routine skills without doing any extensive reasoning. In recent years researchers have used this as a guide to formulate behavioral architectures for robot control.

Most of the proposed approaches are variations of the subsumption architecture as introduced by Rodney Brooks (1986) at MIT. This technique was in sharp contrast to the traditional robot architecture approach in which complex models of the environment

were built before planning and executing actions. Originally, the behavioral approach de-emphasized model building to the extreme by having no internal model of the environment at all. However, recent behavioral architectures do have internal models (Gat *et al.*, 1994; Mataric, 1992). Behavioral architectures also possess many sensor-action streams which are executed in parallel while the traditional approach has only a single processing stream. The biggest advantage of behavioral architectures is that they are readily responsive to environmental changes. However, they have not scaled well to more complex problems involving symbolic reasoning.

A small community of robotics researchers is moving towards a three layered hierarchical robot control architecture (Hexmoor & Kortenkamp, 1995). The lowest layer is a reactive control system, such as the subsumption architecture (Brooks, 1986), which is readily responsive to different sensed stimuli. The top layer is a traditional symbolic planning and modeling system, referred to as a *deliberate* layer. Some of the responsibilities of the symbolic layer include managing the consistency and integrity of world model information, and abstracting new information, as well as validating control issues such as controllability, reachability, and observability. The middle layer, which bridges the reactive and symbolic layers, has not yet been clearly specified. Cognitive psychologists have also noticed different layering based on time scales in human behavior (Hexmoor & Kortenkamp, 1995), as shown by the four distinct layers proposed by Newell (1990): (1) *biological* (on the order of 1 to 10 msec); (2) *cognitive* (on the order of 100 msec to 10 sec); (3) *rational* (on the order of minutes to hours); and (4) *social* (on the order of days to months). Time scale is one way of differentiating the reactive and symbolic layers. The reactive layer is in a real-time feedback control loop, while the symbolic layer is engaged with events that occur at slower intervals. This emphasizes the importance of the two layers being able to function independently of each other. The reactive (i.e., behavioral) layer should be able to guarantee task completion for at least some set of tasks without dependence on the symbolic layer.

There are two ways in which current research has categorized the role of the crucial middle layer. In the first approach, the middle layer is defined as a sequencing layer (Firby, 1987; Gat, 1992; Georgeff & Lansky, 1987; Simmons, 1994). The sequencing transforms a procedural list of task commands into an executable set of reactive skills. The latter react to environmental changes, but are not goal-driven. There is no guarantee that the goal<sup>1</sup> will be achieved. Generally, to guarantee task completion, a planner needs to be integrated with the behavioral controller. In the second approach, the middle layer is defined as a

---

<sup>1</sup>The task command is realized when the goal is achieved.

planner (Arkin, 1990b; Gat, 1992). A robot path, consisting of a set of linear segments, is planned from the start to goal position. Reactive behaviors can override the execution of this plan, but it is not clear how control is subsequently resumed by the path executor. The second approach addresses the requirement for integrating a planner with the behavioral controller, but in this case, the interaction between the planner and behavioral controller could lead to situations where the two levels are not synchronized. The two components need to be intimately interwoven in order to be synchronized. Such a configuration would permit the behavioral controller to execute many tasks independent of a reasoning module. The planner helps secure a guarantee of task completion (i.e., achievement of the goal) in many situations. Under these circumstances, a reasoning system could easily be integrated in the role of an advisor.

This thesis proposes a real-time, asynchronous and parallel distributed robot architecture called SPOTT<sup>2</sup> which bridges the gap between the low and top levels. SPOTT is not a completely distinct layer, but instead integrates behavioral (i.e., reactive) control, planning and some symbolic reasoning into a cohesive system. In essence, SPOTT is a merging of the low and middle layers. A control module links sensing to acting in a classical control feedback loop. Another component is a map database which contains an internal model of the environment. The control module not only controls the actuators, but also updates and maintains the map database. A local planning module makes action decisions based on the currently stored map. A global planning module performs classical Artificial Intelligence (AI) planning based on a search through a graph structure representation of the map. The global path planning module can be replaced by a symbolic reasoner<sup>3</sup> in order to perform more than just AI search. The components of SPOTT are illustrated in Figure 1.1. Many navigational tasks can be performed and guaranteed successful completion with SPOTT, but task completion in certain scenarios will require additional - more complex - reasoning capabilities.

The problem addressed by this thesis is autonomous navigation in a dynamic unpredictable environment which is not completely specifiable by a map beforehand. The three fundamental mobile robot navigational problems are (Leonard & Durrant-Whyte, 1991): (1) “Where am I going?”; (2) “How am I going to get there?”; and (3) “Where am I?”. SPOTT

<sup>1</sup>The reasoning system will provide pertinent information to the planning and behavioral modules, as well as maintain consistency and integrity of world model information. Another role of the reasoning system is to guarantee task completion in the situations where the combined planner and behavioral controller cannot.

<sup>2</sup>A System which integrates Potential fields for planning On-line with TR+ program control in order to successfully execute a general suite of Task commands.

<sup>3</sup>The COCOLOG (Caines & Wang, 1995) reasoning system being developed by Caines and his group at the Centre for Intelligent Machines (CIM) is the one planned to be interfaced with SPOTT.



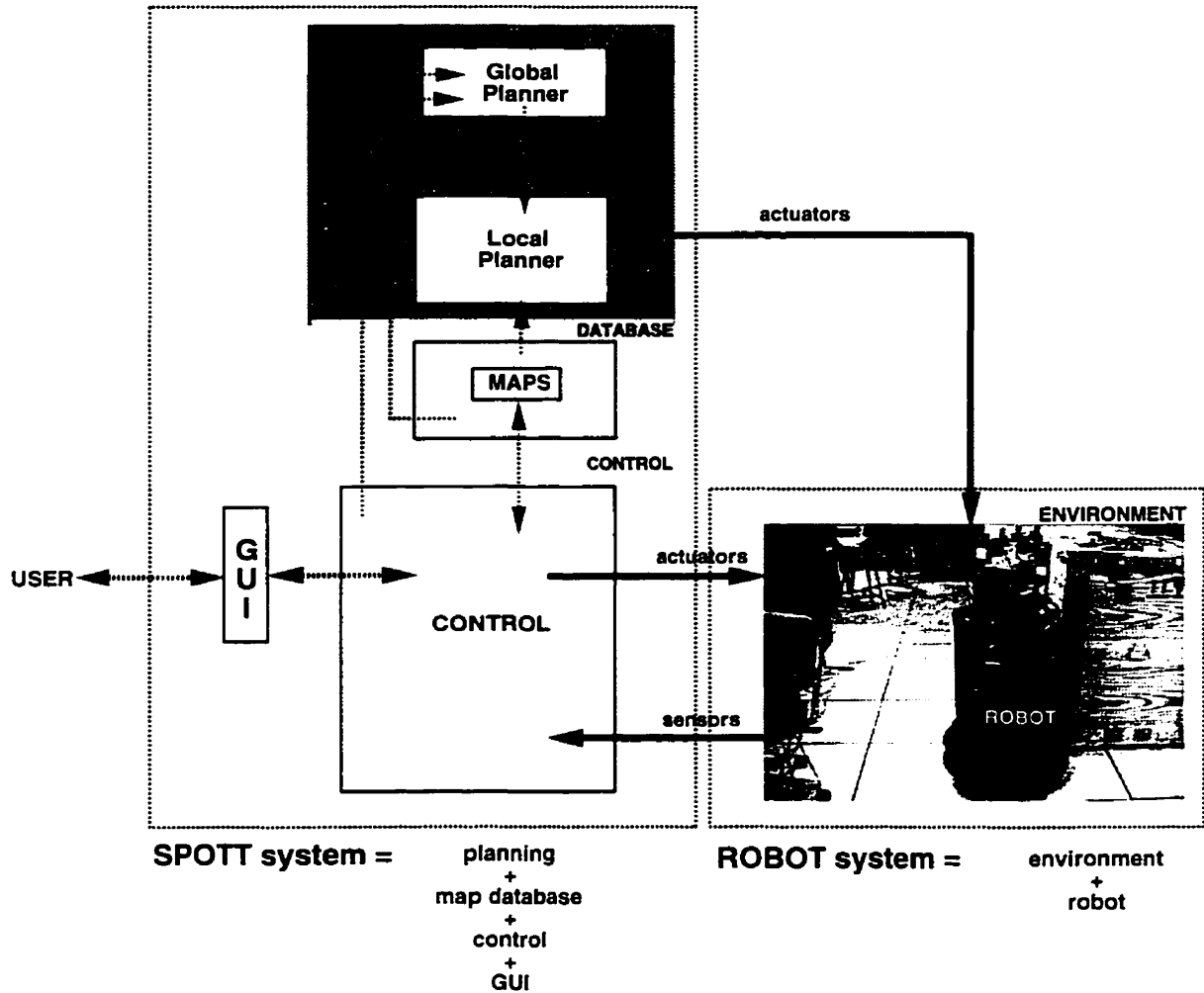


FIGURE 1.1. The SPOTT System. SPOTT consists of a control module, 2 planning modules, an internal map database, as well as a Graphical User Interface (GUI). The control module as well as the local planner form two basic control feedback loops. The dark lines are control flow, whereas the dashed lines are data flow.

addresses the first two questions and the issue of robot position estimation is handled by integrating an existing localization module (Mackenzie & Dudek, 1994) into the system. Under most circumstances, the three problems presuppose that the issue of representation and content of an internal model (i.e., map) has been already addressed. Sensed environmental information (i.e., map) needs to be stored in order to perform navigational tasks such as path planning, obstacle avoidance, and pose estimation. The representational form of the map determines the type and amount of information accessible to the computations associated with the three fundamental navigational problems.

The environment is unstructured, and there is either no a priori information (i.e., map) or there is a partial map of the permanent structures. A “*partial map*” is a map of the

permanent fixed structures in the environment, as would be presented by an architectural CAD drawing<sup>4</sup>. The assumption is made that there is an abstract graph representation of the map available consistent with the architectural CAD map. The abstract graph consists of nodes and edges, where nodes represent rooms or portions of hallway and the edges represent access ways (i.e., doors) between the nodes. SPOTT uses a CAD map of an office and laboratory environment as a priori information. SPOTT has been tested in this environment with a limited set of general navigational tasks. It has been implemented on a collection of SGI and SUN workstations and tested with a Nomad 200<sup>5</sup> mobile robot. The type of sensors on the robot include sonar, infrared proximity, bumper, and two controllable range sensors mounted on pan-tilt heads (QUADRIIS) (Bolduc, 1996).

---

<sup>4</sup>Architectural drawings are readily available in computer readable CAD formats.

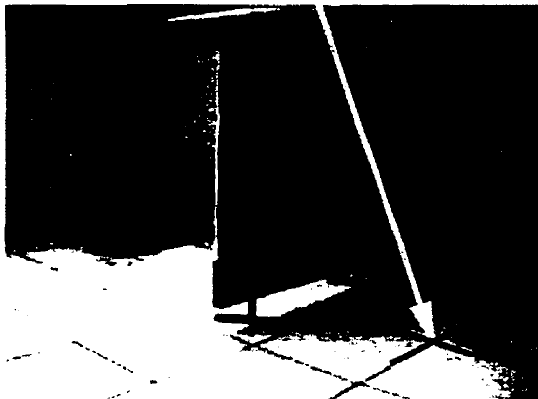
<sup>5</sup>The Nomad 200 is manufactured by Nomadic Technologies Inc., 2133 Leghorn Street, Mountain View, CA 94043-1605, tel. 415-988-7200, e-mail: nomad@robots.com

## 1. Types of Navigational Tasks

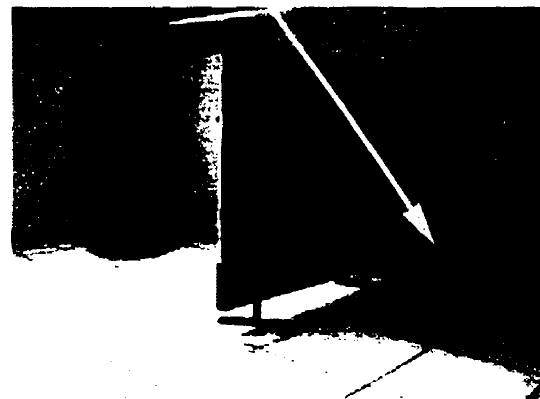
The types of navigational tasks studied in this thesis are based on a language lexicon which is a minimal spanning subset for human 2D navigational tasks (Landau & Jackendoff, 1993; Miller & Johnson-Laird, 1976). User-specified commands and internal communications are formulated using this lexicon. The tasks are based on the verbs “*GO*” and “*FIND*”, and a mode of operation called *intelligent tele-operation*. A task command formulated with the verb *GO* assumes that the goal is a known spatial location, whereas a task command formulated with the verb *FIND* assumes that a description of the object is known but its spatial location is not (see Figure 1.2). *Intelligent tele-operation* is concerned with navigating the robot in a specific direction with no pre-defined target - following simple directional commands such as *forward*, *left*, to name a few - until an obstacle is encountered.

**GOAL: spatial location known**

**GOAL: spatial location unknown,  
but object description  
known (chair)**



**a) GO**

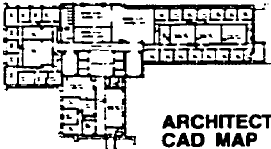








**b) FIND**

**FIGURE 1.2. Types of Navigational Tasks.** The types of tasks considered are based on the verbs *GO* and *FIND*. a) When *GO* is specified, it is assumed that the spatial location of the goal is known. b) When *FIND* is specified, it is assumed that a description of the object sought is known, but its spatial location is not.

The ability of SPOTT to guarantee task completion depends on the type of task and the amount of a priori information. The amount of knowledge SPOTT begins with in order to execute a task, specified by the task lexicon, varies from none to a partial map of the environment. SPOTT can guarantee task completion when the task is *GO* and a CAD map is available a priori, or during the tele-operation mode. In other situations, SPOTT needs to interface to a reasoning module in order to guarantee task completion.

A reasoning module can benefit SPOTT in situations where no initial map is available a priori and during the “*FIND*” task. See Figure 1.3 for a graphical illustration of where a reasoning module could assist SPOTT in order to guarantee task completion. Some of the functions of a reasoning module are to aggregate dynamically sensed map features into abstract symbolic representations, to determine the reachability of the goal location, to provide advice on different search and control strategies, and to perform symbolic graph planning. Currently, a global path planner in SPOTT performs the symbolic graph planning.

<b>A PRIORI INFO</b>  <b>TASKS</b>	 <b>ARCHITECTURAL CAD MAP</b>  <b>A PRIORI MAP</b>	<b>NO MAP</b>
<b>GO</b>  spatial location of goal known	 ● goal reachability	 ● goal reachability ● abstract graph maintenance
<b>FIND</b>  spatial location of goal unknown	 ● goal reachability ● search strategies	 ● goal reachability ● abstract graph maintenance ● search strategies
<b>Intelligent Tele-Operation</b>  move in specified direction until an obstacle is encountered		

What reasoning module could contribute.

-  Full check mark = task completion guaranteed without reasoning  
 Partial check mark = task completion requires reasoning

FIGURE 1.3. What SPOTT Can and Cannot Do. A check mark in a matrix entry indicates the situations where SPOTT can guarantee task completion. A partial check mark indicates that SPOTT cannot guarantee task completion in these situations and the associated caption indicates what role an external reasoning module could fulfill in this situation.

## 2. The SPOTT Robot Control System

The proposed architecture - called SPOTT - consists of (1) a control module, (2a) a local planning module, (2b) a global planning module, (3) a map database (i.e., world model), and (4) a graphical user interface (GUI) (see Figure 1.4). The control module is based on interpreting the task command lexicon within the context of a collection of behavioral decision rules. The behavioral language interpreter evaluates, arbitrates, and executes a collection of behavioral decision rules. In a behavioral decision rule, the antecedent is based on the processing of sensory and world model information, and the consequent is a set of actions. The selected actions control the robot actuators or update the world model. A local planning module continually queries the world model and incrementally calculates a trajectory for locally satisfying the task command goal. The global planning module advises the local planning module on the effects of the global goal. The global planner performs AI search in order to find a path in a symbolic (i.e., abstract) representation of the map. It is not in the critical real-time feedback loop of the system, so it is not subject to the same time constraints as the behavioral controller and local planner. A reasoning module would operate on a similar time scale as the global path planner. The fact that the global planner already exists as part of SPOTT indicates that other time non-critical modules - such as a symbolic reasoning system - could be integrated with SPOTT. The plausibility of integration with a higher level reasoning system shows how SPOTT could provide the crucial link between short-range reaction and long-range reasoning (Hexmoor & Kortenkamp, 1995).

An extension of Teleo-Reactive<sup>6</sup> (TR) programs proposed by Nilsson (1992; 1994) - called TR+ programs - is used for specifying behavioral control in an asynchronous and concurrent implementation. TR programs are an ordered list of production rules which are continually recompiled during execution time into an equivalent hierarchical circuit. A TR program is represented as a graphical tree, called a TR tree, where a condition is specified by a node and an action by an arc. TR+ programs allow condition and action expressions and present the programmer with the ability to control how and when the expressions are evaluated. The formalism permits the "personality" of the robot (i.e., the set of behaviors) to be programmed in a similar fashion to conventional programming (i.e., with parameter passing and binding, hierarchy, and recursion). Potential limitations

---

<sup>6</sup>The actions of the robot are influenced by its goals, hence the term *teleo*. This term was coined by Nilsson in (1992).

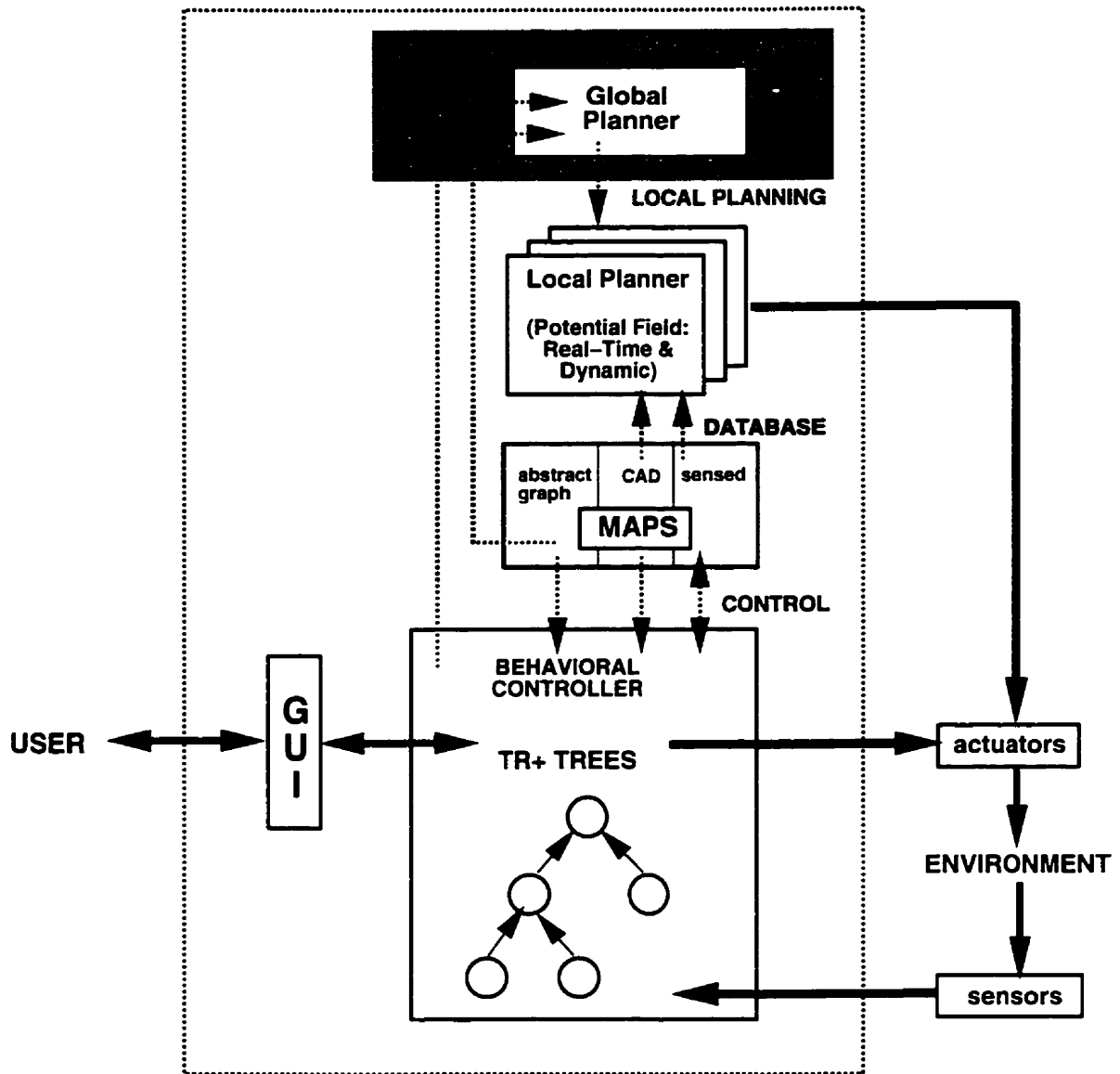


FIGURE 1.4. System Overview. The main components of SPOTT are illustrated. The behavioral controller is an interpreter of control programs. The potential field performs dynamic local path planning. An AI planner produces a path based on an abstract graph representation of the CAD map and provides global information to the local path planner. The map database contains the a priori CAD map, a graph abstraction of the CAD map, as well as a map of the newly sensed features.

with the TR+ formalism arise for problems which are difficult to completely express as a collection of rules, such as obstacle avoidance.

Handling obstacle avoidance within the TR formalism can be difficult since every potential situation would have to be addressed. Many possible contextual rules dealing with obstacle avoidance could actually be encoded as TR+ behavioral rules, but there is no

guarantee that all environmental contexts would be captured. To overcome the combinatorial explosion of obstacle avoidance rules, an *independent* concurrent module performs local path planning based on a potential field technique. The latter is a representation based on a discretized grid of the local map<sup>7</sup> and avoids the combinatorial pitfalls associated with encoding all possible obstacle avoidance situations in a rule-based system. During execution, the TR+ program continually provides goal, obstacle, and robot position estimate information - via the map database - to the potential field module, which in turn is responsible for dynamic local path planning. The map database is comprised of two levels of abstraction. The first level contains the a priori CAD map and a map of all newly sensed features. The second level is a graph abstraction of the CAD map. If the destination is unknown, a TR+ program executes an exploratory set of actions by proposing intermediate goals appropriate for the sensed environment, context, and task.

The local path planner is based on the biologically plausible (Connolly & Burns, 1992) potential field<sup>8</sup> method. It dynamically reacts to the obstacle, goal and robot position estimate information in the map database, which is continually being updated by the TR+ control program. The potential function used by the local planner is a harmonic function, which inherently does not exhibit any local minima. If at least one path exists to a known destination, the path planning strategy is guaranteed to find a path to that goal (Doyle & Snell, 1984). A hierarchical coarse-to-fine procedure involving a collection of harmonic functions at varying resolutions is used to guarantee a timely and correct control strategy at the expense of accuracy. The local path planner gets advice from the global path planner.

The global planning module performs classical AI search - using standard AI graph search algorithms such as Dijkstra's algorithm (Aho *et al.*, 1983) - through a graph structure of nodes and arcs, and advises the local planning module of the local effects of a goal which is outside the current local extent. The two planners - (1) potential field planner (local path planner), and (2) AI-based planner (global path planner) - function in parallel, pass pertinent information between themselves, and operate on different time-scales. The local path planner is in a control feedback loop with the robot and environment, and its actuator commands are crucial to the real-time operation of the robot. On the other hand, the global path planner uses states which change at a slower rate. These are the current and potential physical locations of the robot with respect to the nodes in the graph structure, where a node is a room, or a hallway portion.

<sup>7</sup>The local map contains the features found in a window into the architectural CAD map and the collection of newly sensed features.

<sup>8</sup>The potential field is a field of gradient vectors computed over a discretized version of a potential function.

The global path planner only performs one of the functions of a reasoning system. One can easily envision replacing it by a more comprehensive reasoning system. Such a system would provide additional capabilities such as determining goal reachability, spatial reasoning and map maintenance, and the determination of new TR+ programs (i.e., control strategies). Presently, the local and global path planners, as well as the behavioral controller, function concurrently and communicate asynchronously amongst each other, in order to execute a wide variety of navigational tasks presented by the user.

SPOTT is implemented in parallel, thus permitting real-time asynchronous functionality. The autonomous robot communicates via a radio link to SPOTT which runs on a computer network. SPOTT's processing is distributed across a collection of SUN and SGI workstations. The software tool PVM<sup>9</sup> (Parallel Virtual Machine) (Geist *et al.*, 1993) is used to distribute the control, planning, and graphical user interface processing across a collection of existing processor resources. PVM is a message passing library which allows the harnessing of a collection of heterogeneous processors into a single transparent, cohesive and unified framework for the development of parallel programs. The PVM system transparently handles resource allocation, message routing, data conversion for incompatible architectures, and other tasks that are necessary for operation in a heterogeneous network environment. PVM offers excellent price-performance characteristics compared to massively parallel processors (Sunderam *et al.*, 1993). The portability and heterogeneous property of PVM makes it possible to transfer this architecture on-board the robot in the future. The computation for both the TR+ program control and local planning is also distributed to provide real-time response. Typically, ten to fifteen processors are used in an experiment.

---

<sup>9</sup>This is an ongoing project carried out by a consortium headed by the Oak Ridge National Laboratory.



### 3. Claims of Originality

This thesis addresses the problem of autonomous navigation of a mobile robot in an indoor environment, such as an office or laboratory space. The navigational tasks are based on a language lexicon within two contextual settings: (1) a partial map of the environment is available a priori; and (2) no map is available a priori. A robot control architecture called SPOTT is proposed and implemented as a real-time and parallel system of concurrently executing and co-operating modules. Inherently, the control system is a real-time Artificial Intelligence (AI) system which is responsible for dynamically adapting to changing environmental circumstances in order to successfully execute and complete a set of navigational tasks for an autonomous mobile robot. SPOTT consists of a behavioral controller, a local dynamic path planner, and a global path planner, as well as a map database and a graphical user interface. The SPOTT architecture provides a framework for inclusion of additional sensors and associated perceptual processing algorithms, actuators and control strategies.

SPOTT is a novel robot control architecture because it proposes a way of linking behavioral (i.e., reactive) and symbolic control. The exploitation of existing computational resources by the distributed implementation of SPOTT is additionally innovative. Contributions are also made in the following three areas:

- (i) The Teleo-reactive (TR) control (Nilsson, 1992) formalism forms the centerpiece for the behavioral controller. This thesis has contributed to TR behavioral control research in the following manner:
  - This thesis is the first research work to design, implement and test an on-line and distributed TR interpreter - of concurrently executing behaviors - to handle real-time robot control of an autonomous robot.
  - The basic TR language is extended to handle multiple goals, concurrent actions, and conditional expressions. The extended formalism is called TR+.
- (ii) Path planning is concurrently performed at two levels of abstraction. The local path planner is a potential field approach based on a harmonic function. It is guaranteed to find a path to a goal if such a path exists. The local path planner is in the critical real-time control feedback loop with the environment. The global planner plans a path based on a graph abstraction of the environment. The contributions made to the field of path planning, in particular, the potential field approach using harmonic functions, are as follows:
  - In the local path planner, path computation and execution are done concurrently. In order to guarantee a correct control strategy during concurrent plan

computation, a hierarchical coarse-to-fine procedure based on a set of harmonic functions at varying resolutions is proposed.

- A method is proposed for addressing the issue of planning for global goals when the extent of the potential function is limited. This is necessary because (1) the potential function is a rapidly decaying function which is not computable for all grid elements in a large array, and (2) the computation time increases proportional to the number of grid elements. Thus, the local path planner needs to receive *global goal* information from the global path planner. This assists in sliding the boundaries of the potential function when the robot moves outside the current local extent.
  - A method for computing the harmonic function in real-time and in parallel with existing computational resources is proposed, implemented and tested.
- (iii) There has been very little research done pertaining to human-machine natural language interaction and communication in the field of autonomous mobile robot navigation (Lueth *et al.*, 1994). This thesis is the first to propose and put to use a mobile robot task command lexical subset - consisting of verbs and spatial prepositions - , that is a minimal spanning basis set for human 2D navigational tasks (Landau & Jackendoff, 1993; Miller & Johnson-Laird, 1976). A procedure for quantifying the task command for execution in the behavioral controller and dynamic local path planner is presented. The quantification of the spatial prepositions is shown to depend on two norms. The two norms are the definitions for the spatial prepositions *near* and *far* in the current context of the environment and task.

#### 4. Reader's Guide

A brief reading of this work should include the following chapters and sections:

Introduction (**Chapter 1**),  
Conclusions (**Chapter 8**).

To gain a more detailed understanding of the research presented in this thesis, it is recommended that the following sections be read in order.

- (i) **Chapter 1** covers the main contributions of the work and provides a general summary of the work.
- (ii) **Chapter 2** is a general review of behavioral robot control architectures, mobile robot navigation, as well as a review of the most commonly used sensors for mobile robot research.
- (iii) **Chapter 3** provides a review of the Teleo-reactive (TR) formalism as introduced by Nils Nilsson, and the extended formalism - TR+ - which enhances the TR formalism.
- (iv) **Chapter 4** discusses local and global path planning. A general review of path planning approaches is given alongside a development of the technique used by the local path planner (i.e., potential field using a harmonic function). The dynamic interaction between the local and global path planners is also discussed. The chapter concludes with a collection of potential research topics which could extend the capabilities of the proposed local path planner.
- (v) **Chapter 5** specifies the task command language lexicon and its derivations, and how it ties in with the SPOTT architecture.
- (vi) **Chapter 6**, discusses how SPOTT is implemented in a distributed fashion. The integration of a graphical user interface, and the programming and visualization tools are also discussed.
- (vii) **Chapter 7** discusses the performance and capabilities of the system and the individual components. The results of experimenting with the system are also discussed.
- (viii) **Chapter 8** is a general discussion of the work, outlining advantages, disadvantages, and future research areas.

## CHAPTER 2

### Background

SPOTT extends the behavioral robot control architecture formalism so that it is able to interface with a symbolic reasoning system. In general, the architecture of a robot control system for autonomous navigation is a classical feedback loop as shown in Figure 2.1. A behavioral architecture is a slight variation of the classical composition and is loosely biologically motivated. Biological creatures apparently execute many tasks in the world without doing any extensive reasoning by using a combination of routine skills. In recent years researchers have used this as a guide to formulate behavioral architectures for robot control.

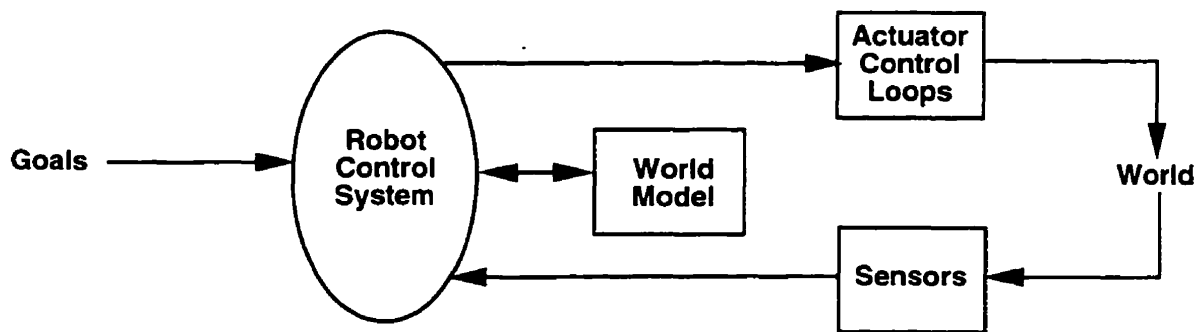


FIGURE 2.1. Simple Robot Control Architecture. All robot control architectures have these basic components, the notable exception being the subsumption architecture (Brooks, 1986) which does not have an internal world model.

## 1. Review of Behavioral Robot Architectures

The basis for most behavioral architecture research efforts has been Brooks' subsumption<sup>1</sup> architecture (see Figure 2.2) (Brooks, 1986). In this architecture, each behavioral decision rule is a simple finite state machine<sup>2</sup>. Each level (i.e., behavior, finite state machine) consists of modules which are complete and self-contained control systems using various types of perceptual information to generate commands for actuators. Each behavior achieves some task, and as new behaviors are added to the system, the level of competence of the robot increases. The ultimate purpose of the robot is defined by its higher level goals (i.e., tasks). The robots built by Brooks and his colleagues (Brooks, 1991; Flynn & Brooks, 1988) did not have behaviors more complex than the wander behavior, contrary to what was originally intended (see Figure 2.3).

The subsumption architecture does not have any memory<sup>3</sup> and thus is prone to possible inescapable cyclic behavior. Therefore, task completion cannot be guaranteed. Various robots have been developed which consist of a few behaviors arranged in a hierarchical order (Brooks, 1991; Flynn & Brooks, 1988). The results are impressive, given the simplicity of the architecture and the lack of an internal representation (i.e., memory) of the external world. The subsumption program is fixed for a particular task and cannot be generalized for a collection of tasks. It has never been scaled up to a level which included behaviors that built and used maps. Its goals are embedded in the higher level behaviors. However, to achieve generality, a goal would need to be a programmable input (Hartley & Pipitone, 1991; Maes, 1991).

Brooks (1991) has strongly advocated the extreme position of having no explicit internal representation of the world. In this case, the external environment acts as the model for the actions. Recently, various researchers (Hartley & Pipitone, 1991; Gat *et al.*, 1994; Connell, 1990; Mataric, 1992) have suggested relaxing this constraint of no internal state for behavioral architectures and have insisted on having some minimal internal state (i.e., memory, maps).

The behavioral architecture contrasts sharply with the traditional robotics approach (see Figure 2.4) (Nilsson, 1969; Moravec, 1983), which involved building complex models of the environment before planning and executing actions. On the other hand, a behavioral

<sup>1</sup>The subsumption architecture was not novel in the 1980's. More than thirty years before, Niko Tinbergen (1951) - one of the initiators of the ethology (ethology is the biological study of behavior) movement - proposed a similar model for animal behavior.

<sup>2</sup>A finite state machine is a structure in which computation is modeled as the transition from one of a finite number of states to another.

<sup>3</sup>Also referred to as internal state. The contents of the memory are a model (i.e., representation) of the world.

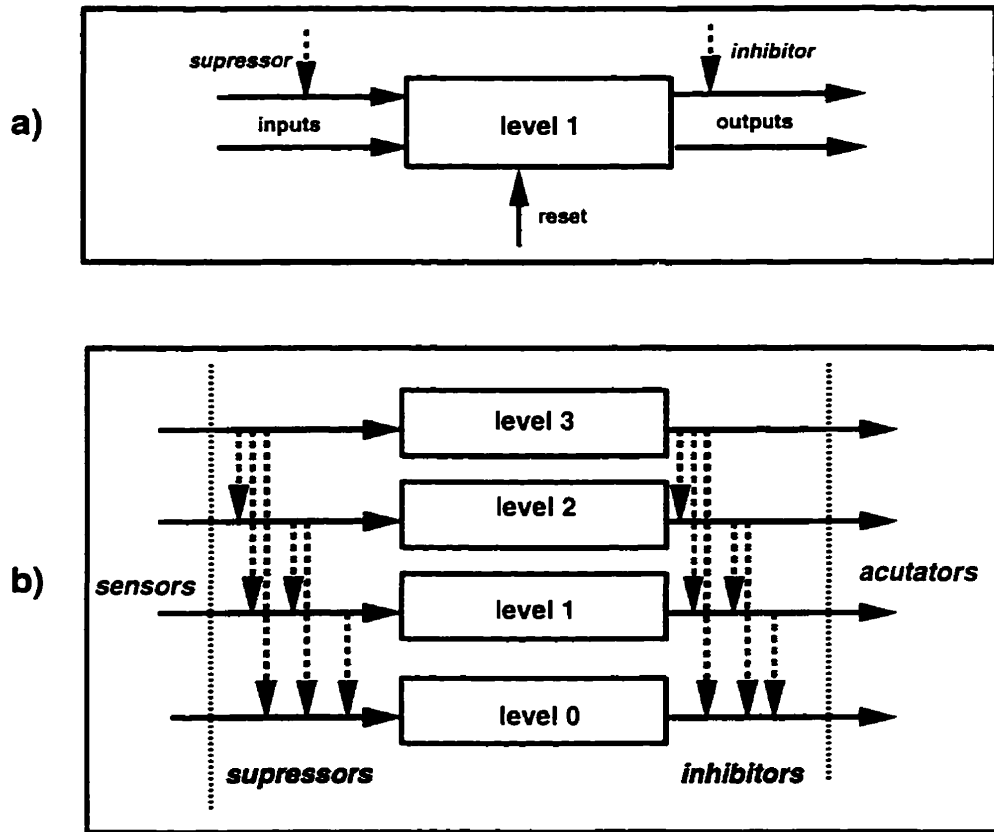


FIGURE 2.2. Brooks Subsumption Architecture: The subsumption architecture is a simple hierarchy of levels (i.e., tasks, layers) where the conditions are continuously computed. The circuitry which implements this architecture is fixed at run-time. (a) A layer has input and output lines. Input signals can be suppressed and replaced by the suppressing signal. Output signals can be inhibited. A module can also be reset to the NIL state. (b) Control is layered with higher level layers (i.e. tasks, behaviors) subsuming the roles of lower level layers when they wish to take control. The dotted lines indicate potential suppressor and inhibitor lines of control. (adapted from (Brooks, 1986))

architecture's actions can be quickly executed after the arrival of sensory data, because the processing involved to make a decision about the next action is usually negligible.

The other significant way a behavioral approach differs from the traditional approach is that a behavioral architecture has a collection of hierarchically ordered streams of action rules (see Figures 2.4 and 2.5), while the traditional approach has a single sensory-to-action processing stream. An action rule is defined as a decision rule based on processed sensor data and world model information. The collection of decision rules and the associated sensory processing are computed in parallel. The rules linking sensing to action are referred to as *behaviors*. When there is relatively instantaneous<sup>4</sup> response, the behavior is referred to as being *reactive*. The behaviors are hierarchically ordered from the most to the least important. The current action to be executed is based on selecting the most important

<sup>4</sup>In the order of milliseconds.

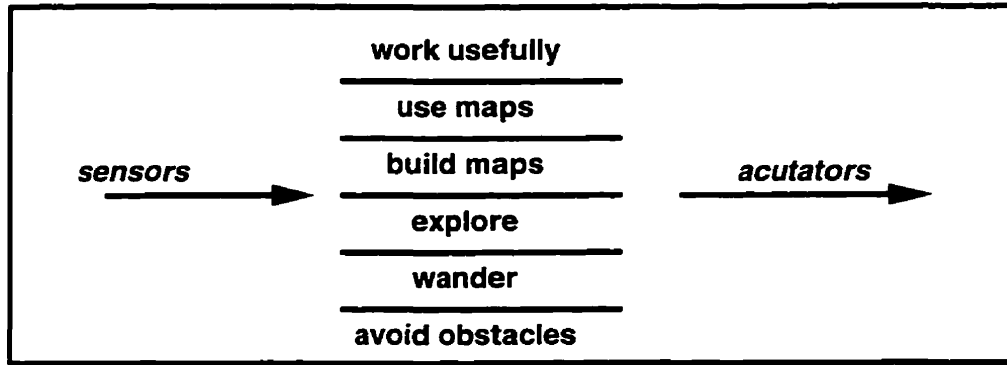


FIGURE 2.3. Brooks' Proposed Hierarchy of Behaviors. Illustrated is a proposed decomposition of a mobile robot control system based on task-achieving behaviors. However, the only levels Brooks and his colleagues ever implemented were the wander and obstacle avoidance levels. (adapted from (Brooks, 1986))

behavior from the collection of behaviors, whose state (i.e., resultant of the executed decision rule) is “on” (or logically TRUE). The action to be executed will either control a robot actuator or update its internal model.

Many other behavioral architectures have been proposed which closely resemble the subsumption architecture (Kadonoff *et al.*, 1988; Kaebbling, 1988; Payton, 1986; Anderson & Donath, 1988; Gat, 1991b; Gat, 1991a; Watanabe *et al.*, 1992; Kweon *et al.*, 1992). The only one which differs significantly from the subsumption architecture is the one suggested by Arkin (1989).

Arkin (1989; 1990a; 1990b) proposed the AuRa architecture (**A**utonomous **R**obot architecture). It is a behavioral architecture based on a dynamic network of active behaviors operating in parallel (as opposed to a hierarchical set of competing behaviors like the subsumption architecture). Rather than driving the actuators directly, there is an intermediate representation, the *potential field*<sup>5</sup> which is a representation for action (i.e., the robot movements). Arkin refers to this type of architecture as a *boiling pot*. A behavior consists of activating a particular *motor schema* depending upon the validity of a particular perceptual input. A *motor schema* is a template, representing an obstacle or goal, which modifies the potential field. There can be many motor schemas activated in parallel. Goal *motor schemas* are attractors, while obstacle *motor schemas* repel the robot. Given that

<sup>5</sup>The potential field is a grid-based method which represents objects of the world as saturated grid elements. The grid elements are either saturated positively (obstacles) or negatively (goals). The space left after the removal of the objects is the free space, and this is the space where forces interact to determine the robot's direction and speed. Obstacles exert negative forces, whereas goals exert positive forces. The forces are summed to generate the potential field. Gradient descent is performed on this surface in order to steer the robot to the goal. The summation of positive and negative forces may create local minima, and thus lead to the robot getting stuck.

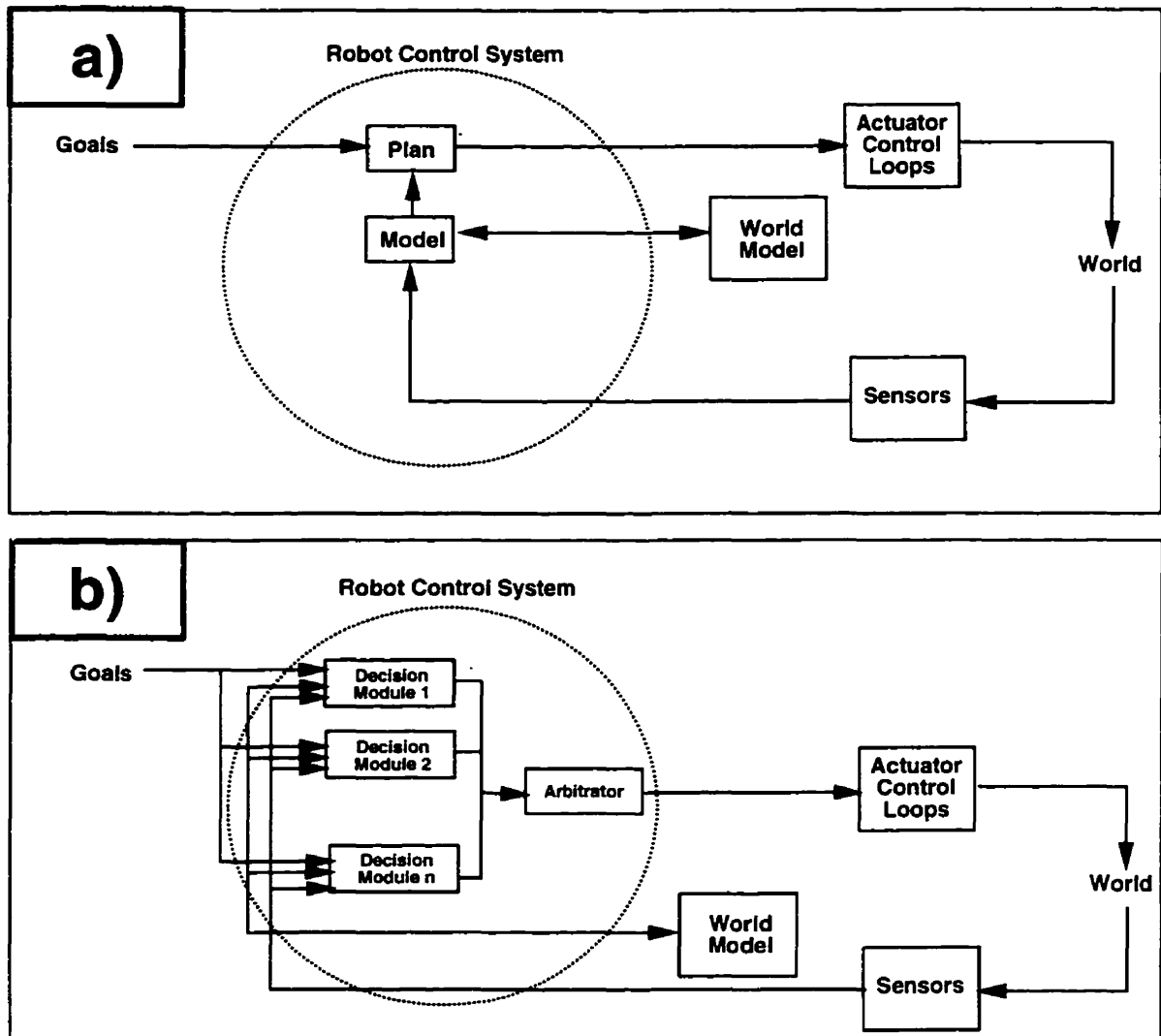


FIGURE 2.4. Traditional vs. Behavioral Robot Control Architecture. (a) In the traditional approach, there is only a single path of control from sensing to actuator activation. (b) In the behavioral approach, there are many paths from sensing to action. Each decision module may include modeling and planning components of varying complexity.

the location of the robot is known, the maximum gradient of the potential field at the robot's current location is used to determine the local direction of the robot. The potential field mechanism that was implemented is problematic in that local minima may be present, which may cause cyclic behavior. Like the subsumption architecture, the AuRa architecture has not scaled up to more complex tasks which could potentially include reasoning.

A small community of robotics researchers is moving towards a three-layered hierarchical architecture (Hexmoor & Kortenkamp, 1995) (see Figure 2.6). The lowest layer is a reactive control system such as the subsumption architecture (Brooks, 1986). The top layer



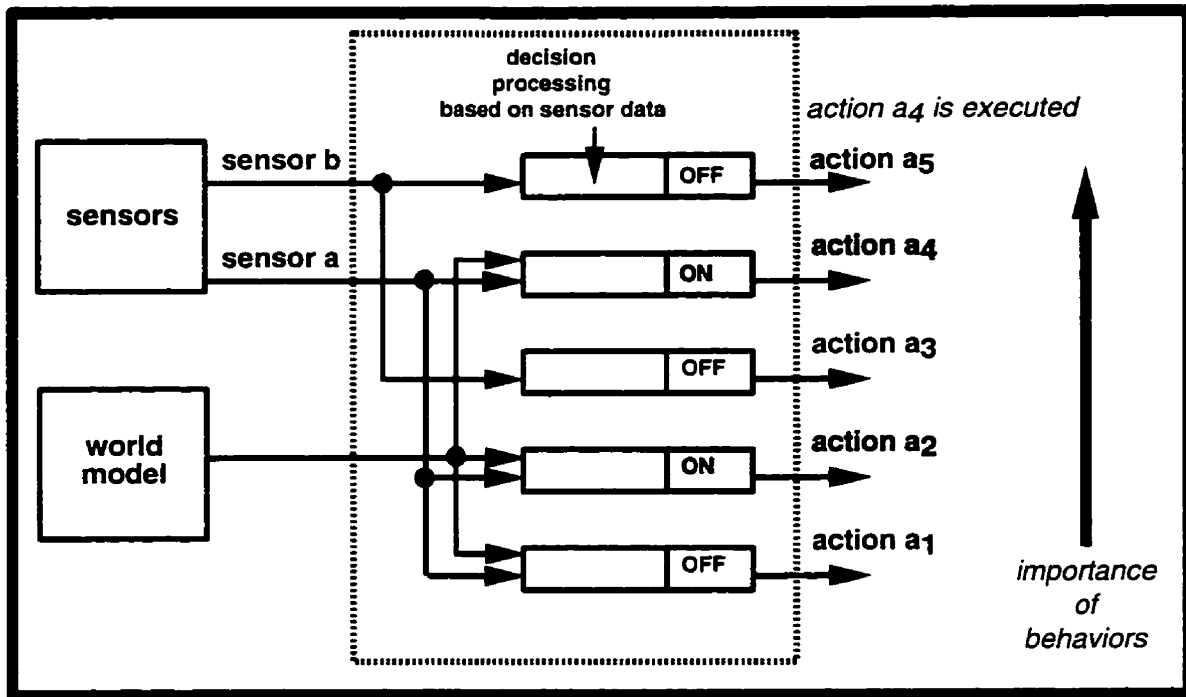


FIGURE 2.5. Behavioral Architecture Example. a) A behavioral architecture has a collection of hierarchically ordered streams of action rules. An action rule implements a decision based on sensory processing and world model information. A specified hierarchical order is the usual arbitration strategy between behaviors. The highest "on" (or logically TRUE) behavior in the hierarchy is the behavior whose action is executed. In this example, action  $a_4$  is executed.

is a traditional symbolic planning and modeling system, referred to as a *deliberate* layer. The reactive and deliberate (i.e., symbolic) layers differ with respect to (1) the type of data used, (2) the speed of reaction to environmental changes, and (3) the type of functionality. Both layers have access to world model information, but the symbolic layer distinguishes itself by only processing the abstract representations. In addition, the reactive layer uses raw or minimally processed sensor input, while the symbolic layer handles highly processed sensor information. Therefore, the reactive layer is readily responsive to different sensed stimuli, while the symbolic layer is comparatively slow. The reactive layer consists of rules which map sensor data to actuator control with a minimal amount of processing. The symbolic layer is responsible for managing the consistency and integrity of world model information, abstracting new information, as well as validating control issues such as controllability, reachability, and observability. The middle layer, which mediates between the reactive and symbolic layers, has not yet been clearly specified. There have been various attempts at defining the middle layer as a sequencing layer, which transforms a procedural list of task commands into an executable set of reactive skills (Firby, 1987; Gat, 1992; Payton,

1986; Georgeff & Lansky, 1987; Simmons, 1994; Ferguson, 1992; Bou-Ghannam, 1992; Connell, 1992). The most popular of the sequencing architectures is the *RAP* (Reactive Action Plans) architecture which has been proposed by Firby (1987). The RAP architecture takes a set of high level tasks generated by a planner and recursively decomposes them into a set of reactive skills. RAPs activate and deactivate these sets of skills, and monitor their execution to see if the robot is being moved towards the goal (Gat, 1992). Similar approaches have been proposed by Payton (1986), Georgeff and Lansky's (1987) *PRS* (Procedural Reasoning System), and Simmon's (1994) *TCA* (Task Control Architecture). The sequencing layer turns reactive behaviors on and off, but it does not guarantee successful completion of any task. Task completion cannot be guaranteed unless a planner is integrated with the behavioral (i.e., reactive) controller.

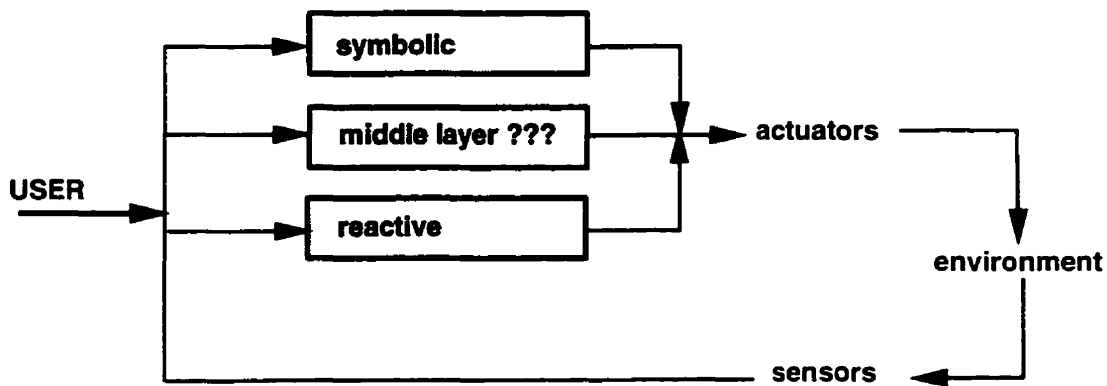


FIGURE 2.6. Three-Layered Hierarchical Architecture. A consensus amongst a small community of researchers applying AI to robotics agree on such a structure for robot control, but there is no agreement on the details of the crucial middle layer.

Some researchers (Arkin, 1990b; Gat, 1992) have suggested defining the middle layer as a planner. In this scenario, a path consisting of linear segments is planned from the start to goal position. Reactive behaviors can override the execution of this plan. However, it is not clear how control is subsequently resumed by the path executor. A path planner is required in order to provide any guarantee of task completion; however, it needs to be intimately interwoven with the behavioral controller. This will permit the behavioral controller to execute many tasks independent of a reasoning module. In this case, a reasoning system could easily be integrated in the role of an advisor, and assist in guaranteeing task completion in the situations where a path planner could not. The reasoning system would provide pertinent information to the planning and behavioral modules as well as maintain internal model consistency.

SPOTT bridges the gap between reactive and symbolic layers by integrating behavioral (i.e., reactive) control and planning into a cohesive system. In essence, SPOTT is a merging of the low (i.e., reactive) and middle layers. It consists of a behavioral controller and a local path planner which operate in a critical real-time feedback control loop with the environment, and a global path planner which functions at a slower time scale. The global path planner's role is to provide crucial global information to the local path planner. The inclusion of a time non-critical module suggests that other such modules (i.e., symbolic reasoning system) could also be integrated. SPOTT can guarantee task completion for many navigational tasks, but more complicated scenarios will require additional reasoning capabilities.

## 2. Interfacing to a Robot Control Architecture

The design of a mobile robot control architecture is influenced by the characteristics of the components it interacts with, particularly: the role of the operator, the complexity of the environment and the capabilities of the mobile robot (see Figure 2.7). The complexity of the control system is inversely proportional to the amount of decision-making initiated by the operator. The type of environment (e.g., static or dynamic) motivates what control strategies to adopt. The mobile robot's actuators and sensors place constraints on the controllability of the robot. For example, the robot's ability to react to changes in the environment is limited by what it can sense.

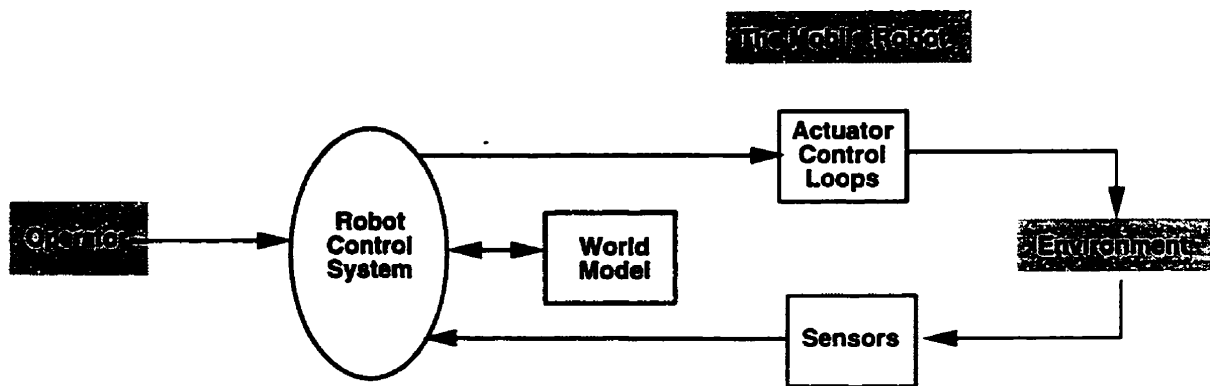


FIGURE 2.7. Interfaces to a Robot Control Architecture. The robot control architecture is influenced by the role of the operator, the complexity of the environment, and the capabilities of the mobile robot.

### 2.1. Role of the Operator

There are two levels of autonomy that robot control can have in relationship to a human operator (Iyengar & Kashyap, 1989).

- **Tele-operation** is the extension of a person's sensing and manipulating capabilities to a remote location. The human operator acts as a supervisor, intermittently communicating to a computer information about goals, constraints, plans, contingencies, assumptions, suggestions, and orders relative to a limited task. The human operator gets back information about accomplishments, problems, concerns, and if requested, sensory data. The robot performs the tedious (i.e., routine) functions and leaves the cognitive tasks and difficult decisions to a human operator.
- The operator supplies a single high level command for **intelligent autonomous robots**. An autonomous intelligent robot is a flexible machine system that can

perform a variety of tasks under unpredictable conditions. It is able to operate without human intervention for extended periods of time.

Between the two, the more interesting and challenging is *intelligent autonomous robot* control, which is addressed by this thesis. As with tele-operation, the human operator gets feedback about the progress of the task, and whether the task has been completed. Contrary to tele-operation, the human operator only needs to issue a single high level command to execute a task. Such is the case with SPOTT, where tasks are specified by a task command language lexicon based on a minimal set of navigational verbs (Miller & Johnson-Laird, 1976) and a minimal spanning subset of spatial prepositions (Landau & Jackendoff, 1993).

## 2.2. The Complexity of the Environment

The environmental complexity can be classified along the following dimensions:

- The environment may be completely known, partially known or completely unknown beforehand. It is very difficult and unrealistic to completely know the robot's environment beforehand, and therefore only the last two cases are truly relevant. The first case is where a partial map of the environment is available a priori. An architectural drawing in CAD format exists for most indoor environments and this can provide a priori information for the robot. The permanent fixed features of the indoor structure, namely the walls, are stored in the CAD drawing, but are never spatially perfect in size or position. Sensory feedback is necessary to validate the CAD map. Additionally, the robot needs to discover all movable features which are not shown in the CAD map (see Figure 2.8). The second case is where no map information is available a priori. Here, the robot needs to reveal both the permanent and movable features.
- The type of environment may be indoor, outdoor, industrial, planetary, or some other, typical of mobile robot applications. The environment can also be evaluated with respect to its complexity: the number of objects, their size, and their spatial layout.
- The objects in the environment may be static or dynamic. The trajectory and speed of the objects are also factors.

Environmental complexity can be reduced significantly by modifying the environment to include artificial landmarks or tracks for the robot to follow. This thesis avoids making such compromises and tackles the more difficult problem of autonomous robot navigation in an unmodified environment.

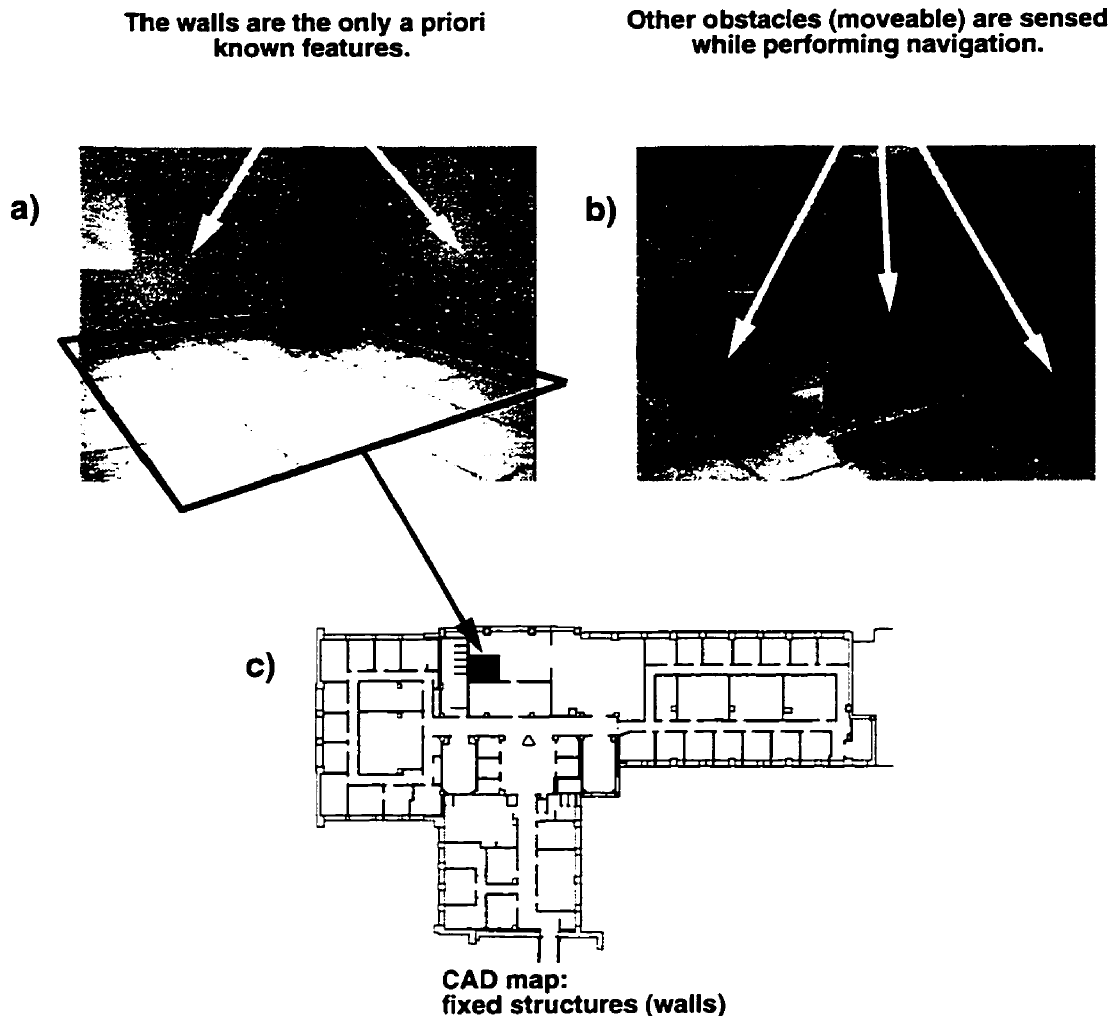


FIGURE 2.8. **Environmental Complexity.** (a) The permanent structures in the environment can be obtained from readily available architectural CAD maps (see (c)). (b) Movable objects must be sensed and noted in the internal model as the robot performs its navigational tasks.

### 2.3. The Mobile Robot

The capabilities of a control architecture are limited by the type of robot and its sensors. The proposed robot control architecture - SPOTT - has been tested using a Nomad 200<sup>6</sup> mobile robot. This vehicle has two degrees of freedom in mobility - translation and rotation - in a two-dimensional plane. In order to execute a movement, the platform first rotates, and then the robot is translated the desired amount in a forward direction. The Nomad

<sup>6</sup>The Nomad 200 is manufactured by Nomadic Technologies Inc., 2133 Leghorn Street, Mountain View, CA 94043-1605, tel. 415-988-7200, e-mail: nomad@robots.com

200 mobile robot base is a three-wheel synchronous non-holonomic system. Its maximum translational speed is 51 cm per second, and the maximum rotational speed is  $60^\circ$  degrees per second. The Nomad 200 has a diameter of 46 cm (53 cm including the bumper), and a height of 83 cm (137 cm including a mounted laser sensing unit). The robot's high centre of gravity does not permit safe passage on inclines. There is always the potential danger of the robot tipping, so the use of the robot is confined to flat terrains. The Nomad is powered by a package of five removable batteries.

It comes equipped with a ring of 20 tactile sensors (pressure sensitive), a ring of 16 ultrasonic sensors, and a ring of 16 infrared proximity sensors. A system of two range sensors (BIRIS) (Blais *et al.*, 1991) mounted on pan-tilt heads, called QUADRIS (Bolduc, 1996), has also been added. QUADRIS is mounted on the top of the basic Nomad platform. Sonar and QUADRIS are used for mapping. Furthermore, QUADRIS is specialized for object recognition. The bumper and infrared sensors are primarily used for safety and accomplish this by mapping very close objects which may be missed by the other two sensors. These sensors are different from sonar and laser because their data is readily available and requires minimal analysis. Figure 2.9 illustrates the placement of the sensors on the Nomad 200 mobile robot. The sensors are of the type most commonly used in the mobile robot literature.

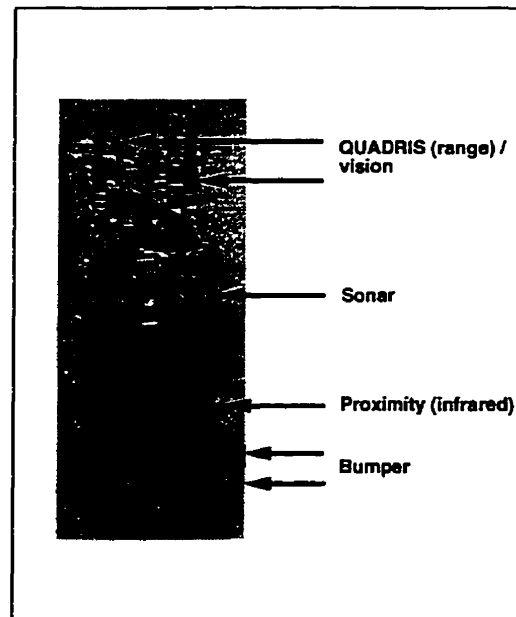


FIGURE 2.9. Mobile Robot Sensors. The sonar, infrared and bumper sensors come with the original Nomad 200. A pair of range sensors mounted on pan-tilt heads - QUADRIS - is also installed.

Most common sensors (Luo & Kay, 1989; Harmon, 1987a; Everett, 1995) installed on mobile robots are associated with a variety of problems when used as the sole sensing device. Therefore, in practice, it is common to use many types of sensors to compensate for each sensor's inadequacies.

### 2.3.1. Vision

Visual sensors are considered passive since they only receive information about the environment and do not emit any signals. One of the major problems with using vision is that it lacks explicit information regarding the third dimension (i.e., depth). Methods for determining depth from video cameras include *depth from stereo disparity* (Longuet-Higgins, 1981; Prazdny, 1980; Marr & Poggio, 1976), *depth from camera motion* (Matthies *et al.*, 1989), *depth from focusing* (Krotkov, 1989b), *depth from texture gradients* (Witkin, 1981), *depth from shading* (Horn, 1977), *depth from occlusion cues*, and various combinations of these methods. Video cameras are rarely used by mobile robots for navigational control because of the large amount of processing required<sup>7</sup> to compute depth information which makes real-time operation very difficult. However when they are used, the most popular method for extracting depth information is based on the *depth from stereo disparity* technique. This was used to navigate the Stanford Cart (Moravec, 1983)<sup>8</sup>, but the robot moved only 1 m every 10 to 15 minutes. There was also significant error in position (i.e., 20 %) of the object features sensed in the environment. Current methods (Ezzati, 1995) for extracting *depth from stereo disparity* are approaching real-time speeds, so that the use of video cameras may soon become feasible. Besides navigation, visual sensors are also useful for recognition tasks: finding a particular object or landmark recognition (Krotkov, 1989a).

### 2.3.2. Acoustic (Sonar)

Sonar sensors are a type of active range sensor and function by emitting sound waves and measuring the time delay it takes for the echo to return. A single transducer is often used as both the transmitter and receiver. The distance to the object reflecting the pulse can be inferred from the knowledge of the time taken for the sound to return and of the speed of sound (approximately 0.3 m/ns). A typical ultrasonic ranging device operates with frequencies of roughly 50 kHz (Dudek *et al.*, 1993). Sonar units are usually installed on a robotic platform in one of two ways: (1) a single sonar unit is mounted on top of a

<sup>7</sup>In general, visual sensors are critically dependent on ambient lighting conditions. The analysis of a visual scene as well as the associated registration procedures can be complex and very time-consuming.

<sup>8</sup>The Stanford Cart project was conducted at Stanford University AI Lab between the years 1973 and 1980.



rotating platform which points it in different directions<sup>9</sup>; and (2) a collection of sonar units are uniformly distributed around the exterior of the robot. There is always the problem of crosstalk with multiple sonar units. This is especially true in cluttered environments, where sound waves can reflect from multiple objects and then be received by other sensor elements. To compensate for crosstalk, repeated measurements are often averaged to bring the signal-to-noise ratio within acceptable levels, but at the expense of the additional time required to determine a single range value (Borenstein *et al.*, 1996). Other problems characteristic of *acoustic sensors* include very low resolution, large specular reflections (i.e., multiple reflections), slow speed, fuzzy images, temperature (and to a lesser extent humidity) dependence, and multiple echoes, as well as problems with the physics of the sonar process<sup>10</sup>. Sonar sensors are useful because they are relatively inexpensive, fast and provide range data in all directions simultaneously. Sonar systems are less expensive than other types of range sensors (i.e., laser rangefinders) because of the relatively slow speed of sound in air (compared to light), which places milder timing demands on the required circuitry.

The typical operating range of sonar sensors is 10 cm to 914 cm (Borenstein *et al.*, 1996). In cluttered environments, reliable measurements can only be obtained from 2 or 4 m (Dudek *et al.*, 1993) away. Sonar sensors have been used for mapping (i.e., creating an internal model of the environment) tasks (Crowley, 1985; Bozma & Kuc, 1991; Elfes, 1987; de Saint Vincent, 1986) and for estimating the position of the robot (Drumheller, 1987; Leonard & Durrant-Whyte, 1991; Mackenzie & Dudek, 1994).

### 2.3.3. Laser Rangefinders

Laser rangefinders are another type of active range sensor. They project a structured illumination pattern of their own, and make use of either the *geometry* or *time of flight* to determine the range of objects. *Time of flight* measurement determines the distance of an object by measuring the time it takes for light to travel from the source to the object and back. Triangularization (i.e., *geometry*) systems broadcast a beam of light with a known shape (e.g., point, line, cross), and the camera observes reflection or scattering from the object to determine depth based on the geometry. The *time of flight* method is fine for the outdoors, but for indoor use, the size of the sensor head and power consumption are critical factors (Harmon, 1987a). The *triangularization* technique uses a single camera and usually a line projector. It is not recommended for long distance measurements and, just like sonar,

<sup>9</sup>The robot manufactured by Cyberworks Inc. uses this sonar arrangement (Cyberworks Inc., 31 Ontario Street, Orillia, Ontario, L3V 6H1).

<sup>10</sup>The sonar chirp is not an infinitely narrow beam of infinite power but has a finite angular extent with a complex cross-sectional energy distribution (Dudek *et al.*, 1993).

sometimes gets confused with reflections (Harmon, 1987a). In general, laser rangefinders require an intense energy source, and they have a short range and slow scan rate<sup>11</sup>. Laser rangefinders may avoid all the drawbacks of visual and acoustic-ranging systems but this is often at the expense of cost and potential hazard to humans. Laser rangefinders have been used for estimating robot position (Cox, 1991; Moutarlier & Chatila, 1990) using a method which correlates laser range data with an existing map.

The BIRIS<sup>12</sup> Range Sensor (Blais *et al.*, 1991) is a compact optical 3-D range sensor developed by the NRC (Canada's National Research Council), which is based on the use of a double aperture mask in place of the diaphragm of a standard camera lens. A pair of overlapping stereo images are produced on a single film plane. If a laser line is projected perpendicular to the line connecting the two irises, then stereo correspondence is simplified. This laser line, when viewed through the double aperture, produces two lines whose separation correlates with the *range distance*. The BIRIS range sensor is able to detect objects up to 5 m away. A single scan range estimation can be computed with a simple algorithm which permits real-time computation (i.e., 10 frames per second). In order to provide a full range image, the sensor must be swept across the scene. The QUADRIS (Bolduc, 1996) system uses two BIRIS sensors mounted on pan-tilt heads.

#### 2.3.4. Other Types of Sensors

Other types of sensors used by mobile robots include contact and proximity sensors. Obstacle detection based on just contact sensors limits the speed of the robot because contact must be made before detection can take place. Thus the introduction of proximity sensors which are based on LED (Light Emitting Diode) phototransistor pairs. LED's function by emitting light into the environment, and receiving light reflected by objects. Usually they can measure range up to 45 cm away. Contact and proximity sensors are mostly used for safety monitoring<sup>13</sup>.

Most mobile robot designs incorporate many sensors (Kriegman *et al.*, 1989; Yuta *et al.*, 1991; Evans *et al.*, 1992; de Saint Vincent, 1986) to overcome the shortfalls of any individual one. There are many practical considerations when deciding which sensors to use, such as cost, power consumption, speed, resolution, just to name a few. Many mobile robots use

---

<sup>11</sup>For more information about laser rangefinders and processing techniques, see (Everett, 1995) and (Jarvis, 1983). Two typical laser rangefinders (both are 'time of flight' (tof) sensors) are as follows: (1) SEO AutoSense (9 m maximum range, 1% accuracy, 29.3 rps scan rate, weight of 5 lbs.); (2) RIEGL LSS390 (range of 1 to 60 m, 10 cm accuracy, 10 scans/sec, weight of 4.86 lbs.) (Everett, 1995).

<sup>12</sup>Based on two pinholes or irises, therefore the name Bi-IRIS.

<sup>13</sup>As a last resort, if obstacles are not detected by other longer range sensors.

acoustic or laser rangefinders as opposed to video cameras, because depth information is explicit with these sensors.

The selection of sensors used by SPOTT was based on the availability of the processing algorithms. The LED and bumper sensors require little processing and were integrated for short-range sensing. Sonar and QUADRIS are used for long-range analysis. Mapping (i.e., creating an internal model) and localization (i.e., estimating the position of the robot) software based on sonar data (Mackenzie & Dudek, 1994) had been already developed and research at CIM has just begun in using QUADRIS for mapping and object recognition (Bui, in preparation). Mapping, localization and recognition are the functions required for the navigational tasks SPOTT executes.

### 3. Robot Control Issues

Autonomous robot control is difficult because of the timeliness of decision-making and acting, the unpredictability of the world, and the imperfections associated with sensors and actuators. Behavioral architectures are attractive because they are simple, fast and produce interesting and complex behaviors which appear to be intelligent. An intelligent agent (i.e., the robot) is defined as having the capability to adapt to a changing environment while performing a particular task. The design of a behavioral architecture for mobile robot control has to take into consideration the following issues:

- (i) How well does the architecture **scale** to more complex problems?
- (ii) Is the architecture **general** enough for a wide variety of tasks and environments?
- (iii) Are the executed actions of the robot **predictable** beforehand?
- (iv) Is the operator able to **monitor** the robot's interaction with the environment?

The **scalability** of an architecture is its ability to handle increasingly more complex problems that demand a greater amount of knowledge without changing its underlying mechanisms. A robot's *architecture* should not be confused with a robot's *programming language* (Horswill, 1995). A robot's *programming language* is the means of describing the tasks and the techniques used to solve them, independent of the implementation mechanism. It remains the same from one task to another. On the other hand, a robot's *architecture* is a collection of specific components with fixed connections that can jointly perform a broad range of tasks with minimal modification. Given this distinction, the subsumption architecture is more like a programming language than an architecture. There is no specific set of components that all subsumption robots have in common<sup>14</sup>. In contrast, SPOTT is an architecture<sup>15</sup> with one of its components - TR+ interpreter - interpreting robot control programs. It is actually the flexibility of the programming language which determines the scalability of the architecture. Limitations of the architecture may require the addition of a new component (e.g., reasoning), but this should not affect the existing mechanisms. In other words, provisions for expansion should be allowed for in the original design.

The **generality**<sup>16</sup> of an architecture is a measure of the types of tasks and environments which can be successfully operated upon. The architecture should be able to do various tasks without having to be reprogrammed or physically rebuilt. Brooks' robots were built

<sup>14</sup>Each subsumption robot is built out of a common class of components; all are built from finite-state machines inhibiting one another, albeit different finite-state machines (Horswill, 1995).

<sup>15</sup>SPOTT's modules - TR+ interpreter, local path planner, global path planner - and their interconnections are fixed for all tasks that the system can perform.

<sup>16</sup>*Generality* and *scalability* are not mutually exclusive properties: changing the task or environment may make the problem more complex.

to perform a single task, and were not able to extend performance to a suite of tasks. The single task was hard coded for each robot. Other proposed behavioral architectures do not clearly define what tasks (or sets of tasks) the robot can perform (Arkin, 1990b; Gat, 1992).

The architecture's functionality is **predictable** if it is able to guarantee successful operation and task completion for various tasks and environments. The subsumption architecture (Brooks, 1991) cannot guarantee that the robot will complete the task successfully. This mainly stems from the extreme position of having no internal model. The emerging overt behavior of the robot in most behavioral systems, including the subsumption architecture, is only evident by observing the robot. In order to guarantee task completion, the gross actions<sup>17</sup> should be deterministic, predictable, and repeatable. Behavioral architectures such as the one by Brooks are usually non-deterministic. An internal model and a dynamically adaptable planner need to be integral parts of an architecture in order to provide any task completion guarantees.

Analytic or graphical tools are necessary for **monitoring** (Brooks, 1991), debugging and understanding expected and unexpected interactions with the environment, as well as simplifying the task of programming control strategies. Debugging or the verification of execution can either be done on-line or post-mortem.

Besides interfacing and the general issues mentioned above, the robot control architecture needs to be designed to solve a particular class of problems intelligently and in real-time. SPOTT addresses the problem of *mobile robot navigation*. It quickly reacts to a dynamic environment in order to avoid collision with obstacles while still striving to fulfill its task objective. Real-time and intelligent interaction with the environment is addressed by a relatively new research area called *real-time AI*. Other higher level concerns such as learning and reasoning are beyond the scope this thesis and are left as potential research areas for future extensions to the ideas presented here.

### 3.1. Mobile Robot Navigational Problems

The three fundamental questions (Leonard & Durrant-Whyte, 1991) of mobile robot navigation are: (1) "*Where am I going?*"; (2) "*How am I going to get there?*"; and (3) "*Where am I?*". The questions can be rephrased as follows: (1) goal determination and path planning; (2) path execution; and (3) the ongoing estimation of the robot's position. All of these problems rely on the robot's "*ability to sense the environment and to build a*

---

<sup>17</sup>A gross action is an abstract collection of actions such as obstacle avoidance. This gross action is deterministic if the functionality, such as obstacle avoidance, is guaranteed to be successful. It is difficult to predict the exact trajectory due to the non-repeatability of environmental circumstances.

*representation of it, and to use this representation effectively and efficiently*" (Talluri & Aggarwal, 1993). This introduces the issue of *map building* (i.e., *mapping* of the environment).

### 3.1.1. Path Planning

The future locations of the robot include both the destination and the path to be followed. Path planning usually requires that a map (e.g., a CAD map of the indoors environment) be transformed into a graph structure, which is then searched for a path. This can be achieved by parcelling the map into equally - or unequally - sized convex polygonal regions. Nodes are used to represent the regions, and edges connect nodes that share a common boundary. Each node and edge is labeled as passable or impassable, and a search is initiated to find a path through passable nodes via passable edges from a start to stop node. Path planning for a fully autonomous mobile robot has always been a computational bottleneck (Hwang & Ahuja, 1992) requiring complex search algorithms.

In contrast to autonomously planning a route, tele-operated mobile robots avoid this issue and permit a human to perform high level decisions. In this case, the research effort is directed towards simplifying the decisions an operator must make by providing sufficient and reliable information (i.e., the man-machine interface).

### 3.1.2. Path Execution

The execution of the path involves sensing the environment and making reactive decisions to adapt to a potentially changing environment. There are several ways of executing a path.

- *Preprogrammed paths* can be enforced by either buried wires or painted lines which are sensed. This requires extensive modifications to the environment (Harmon, 1987b).
- *Following structures* in the environment is a simple way of getting around. Examples include wall following and road following. Road following techniques involve following road edges, but certain problems may arise when the road edges can not be detected (Thorpe *et al.*, 1988; Dickmanns *et al.*, 1990).
- *Obstacle Avoidance*: Obstacle avoidance can be performed by either using heuristic reactive responses (Smith *et al.*, 1975; Crowley, 1985; Manz *et al.*, 1991; Waxman *et al.*, 1987; Moravec, 1983; Brooks, 1986; Arkin, 1990b), computationally expensive reasoning (Nilsson, 1969), or dynamic path planning. The latter involves concurrently recomputing and executing the path. Such an approach is proposed in this thesis.

### 3.1.3. Robot Position Estimation

A robot's position can be computed from information about either the robot's motion or the relative positions of external cues for which absolute position is known. Dead reckoning<sup>18</sup> has good short term performance, but the errors are cumulative. Odometry<sup>19</sup> is able to provide the vehicle with an estimate of its position, but the position error grows without bound unless an independent reference is used periodically to reduce the error (Cox, 1991). Inertial systems<sup>20</sup> may be able to provide good position estimation. However errors also tend to accumulate, and the equipment is very expensive (Borenstein *et al.*, 1996). For a wheel-based robot, position error is due to wheel slippage, which is difficult to model. Some factors which influence wheel slippage include the type of floor surface, robot velocity, wheel alignment, and the magnitude and number of performed turns.

There are two basic methods for determining the position of the robot on an ongoing basis. Landmark detection is perhaps the simplest. Installing pre-defined beacons<sup>21</sup> requires modification of the environment and does not lend itself to exploration of unknown terrain. Known landmarks (Evans *et al.*, 1992; Hebert, 1989; Cox, 1991) can be identified and matched to a map when the terrain is known. Landmarks can either be *natural* or *artificial* (Borenstein *et al.*, 1996). *Natural landmarks* are those objects or features that are already in the environment and have a function other than for robot position estimation. *Artificial landmarks* are specially designed objects or markers that need to be placed in the environment with the sole purpose of enabling robot position estimation. When the known landmarks are artificial, they can be placed in positions which are optimal for detectability even under adverse environmental conditions. On the other hand, natural landmarks require no preparation of the environment, but the environment needs to be at least partially known a priori.

The second method is to match sensor data to a known map (Mackenzie & Dudek, 1994; Schiele & Crowley, 1994; Weiß *et al.*, 1994; Rencken, 1994), either geometric or topological. Geometric maps represent the world in a global coordinate system, while topological maps represent it as a network of nodes and arcs. SPOTT uses an existing robot localization

---

<sup>18</sup> *Dead reckoning* is derived from "*deduced reckoning*" of sailing days. It is a simple mathematical procedure for determining the present location of a vessel by advancing some previous position through a known course and velocity information over a given length of time (Borenstein *et al.*, 1996).

<sup>19</sup> This method uses encoders to measure wheel rotation and/or steering orientation. It is also the simplest implementation of dead reckoning.

<sup>20</sup> This method uses gyroscopes and sometimes accelerometers to measure the rate of rotation and acceleration.

<sup>21</sup> The absolute position of the robot is found by measuring the direction of incidence of three or more actively transmitted beacons. The transmitters, usually using light or radio frequencies, must be located at known sites in the environment.

method (Mackenzie & Dudek, 1994) which is based on correlating sonar data with an a priori geometric map.

### 3.1.4. Map Building

There are two common map representations: *geometric* and *topological* (Borenstein *et al.*, 1996). The geometric map can either be a grid map or a map consisting of a collection of geometric entities (i.e., lines, polygons) tied to a fixed coordinate reference system. An example of a grid map is called the *occupancy grid*. It is a probabilistic tessellated representation of spatial information, where each grid entry is updated as more sensor information is made available about that specific region in space (Elfes, 1989). Occupancy grid-based maps require little computation and permit relatively simple integration of different sensors. However, they present difficulties in modeling dynamic obstacles and require a complex estimation process for determining the robot's position (Borenstein *et al.*, 1996). The *topological* approach is based on recording the geometric relationships between the observed features. This produces a graph where the nodes represent the observed features and the edges represent the relationships between them. Topological maps are used primarily for *robot position estimation* and *path planning* (Borenstein *et al.*, 1996). Robot position estimation occurs when the robot is near one of the nodes and is able to match sensor data with the stored map information at the node. Path planning is the selection of a route to a destination location, visiting the necessary nodes along the way.

A map is built by fusing a priori knowledge (e.g., CAD map), different sensory data and perhaps deduced information (i.e., by a reasoning module). Mobile robotics literature (Hager, 1990; Durrant-Whyte, 1988) categorizes the different fusion methods as follows: (1) *competitive*, where the sensors are supplying redundant or competing observations; (2) *complementary*, when the sensors provide unique information which constrains or refines observations from another; and (3) *independent*, as only one sensor contributes at a time. The decision of what type of fusion strategy to use should be based on how the sensory data is being utilized to support the robot's intended action (Murphy, 1996).

SPOTT uses a dynamic path planner based on the potential field technique for determining action. Its map consists of an a priori CAD map fused with sensory (i.e., sonar, range, infrared, bumper) data. The data is fused by using the *competitive* method. When new sensory data arrives, it is compared to the existing map to see if the corresponding spatial locations are already occupied. If it is, then the new sensory data is discarded; otherwise it is entered into the map. The advantage of this method is that it is simple and



adequate for navigation in most circumstances<sup>22</sup>. The disadvantage is that recently sensed redundant data is not used to refine currently stored map features. Uncertainty is addressed by padding the geometric map features by a predetermined expanse in all directions (e.g., a line feature becomes a rectangle).

### 3.2. Real-time AI

An autonomous robot needs to respond quickly and intelligently to changes in the environment. A relatively new area called *real-time AI* (Musliner *et al.*, 1994) addresses the difficult issues which arise when deadline constraints are combined with complexities arising from (1) uncertainty in sensing, environmental modeling, and actuator activation, and (2) limited computational processor resources. *Real-time AI* is the convergence of real-time systems and Artificial Intelligence (AI) methods. A control system is real-time if the reaction rate of control is faster than the rate of change in the environment. An AI method executes search by finding an appropriate set of actions to lead an agent from some initial state to a goal state. The two areas are merging because AI methods are tackling more realistic domains requiring real-time response and real-time systems are moving towards more complex applications requiring intelligent behavior. Building real-time AI robot systems is difficult because of the constraints imposed by the robot: (1) The robot is subject to *bounded computation* because its data processors have limited memory and speed; and (2) The robot is also subject to *bounded reactivity* due to the range and accuracy limitations of its sensors and actuators. There are three ways in which AI can be integrated with real-time systems: (1) AI is embedded into a real-time system; (2) real-time is embedded into an AI system; and (3) a real-time and an AI system co-operate concurrently.

AI is embedded into a real-time system when search-based problem-solving is constrained to meet real-time deadlines. In general, AI tasks, such as planning and search-based problem-solving, are ill-suited to real-time scheduling as the scheduling relies on the worst-case execution times for all tasks, and these tasks often have unknown or extremely large worst-case execution times (Paul *et al.*, 1991). The unpredictable execution time is due to variations caused by search and backtracking operations. There are two approaches for embedding AI into a real-time system. In the first, the execution time variance inherent in search-based AI problem-solving is reduced to make these techniques viable for worst-case scheduling. An example is the purely reactive AI architectures (e.g., subsumption architecture (Brooks, 1986)), where all condition-action rules are assumed to run concurrently.

---

<sup>22</sup>This is not the case for moving objects.

A hierarchical ordering of the rules is used to decide the current active rule without the need for any AI search<sup>23</sup>. Another way of reducing variance is to execute a collection of distinct AI solving methods. Each method requires different computation times depending on the circumstances. The result is obtained from the method which meets the current situation's real-time constraints. In the second approach, the AI task is cast as an incremental and interruptible algorithm. Such algorithms are referred to as *anytime algorithms* (Mouaddib & Zilberstein, 1995). The result of an *anytime algorithm* is available from the start of computation at the expense of accuracy. The accuracy improves gradually, directly proportional to the execution time, to some optimum value.

When real-time is embedded into AI, the overall system employs AI deliberation techniques but under some circumstances these techniques may be short-circuited in favour of a real-time reflexive response. The main problem with this technique is that if reflexive actions can bypass normal deliberation mechanisms, it may be difficult or impossible for deliberation processing to reason about the changes introduced by the reflexive responses.

The third method in which AI is integrated with real-time is to have two independent, concurrently executing, and co-operating processes. Synchronization problems may occur because the AI system operates at a time scale which is much slower than the real-time system.

The approach taken by SPOTT combines all three methods for merging real-time control and AI deliberation. Examples of AI being embedded within a real-time system are the behavioral controller - TR+ program interpreter - and also the potential field local planning mechanism. TR+ programs eliminate search by having a fixed hierarchy of ordered rules. The potential field path planning strategy is an *anytime algorithm* (i.e., the local path is always available). An instance of real-time being embedded into AI occurs when reactive behaviors (i.e., TR+ rules) override the currently planned movements (i.e., local potential field path planner). A shared map database resolves local planning and behavioral control conflicts. In addition, the local path planner can also be viewed as a real-time module interacting with the concurrently executing global AI path planner. The local path planner is in the critical real-time feedback control loop, whereas the global path planner is not.

---

<sup>23</sup> A production system also does not require any AI search.

#### 4. The SPOTT Robot Control System

This thesis proposes an architecture called SPOTT, which provides a bridge for linking behavioral (i.e., reactive) and symbolic control. SPOTT is a real-time AI system for autonomous mobile robot control which is responsible for dynamically adapting to changing environmental circumstances while executing navigational tasks. The architecture consists of (1) a control module, (2a) a local planning module, (2b) a global planning module, (3) a map database (i.e., world model), and (4) a graphical user interface (GUI) (see Figure 1.4). The control module consists of an interpreter executing behavioral (TR+) programs. The selected actions - from the behavioral controller - control the robot actuators or update the world model. A local planning module continually queries the world model and incrementally calculates a trajectory for locally satisfying the task command goal. The local planner and the behavioral controller are in the critical real-time feedback control loop of the system. The global planning module advises the local planning module on the effects of the global goal. The global planner performs AI search to find a path in a symbolic (i.e., abstract) representation of the map. It is not in the critical real-time feedback loop of the system, so it is not subject to the same time constraints as the behavioral controller and local planner. A reasoning module would operate on a similar time scale as the global path planner. The fact that the global planner already exists as part of SPOTT indicates that other time non-critical modules - such as a symbolic reasoning system - could be integrated with SPOTT. The plausibility of integration with a higher level reasoning system shows how SPOTT could provide the crucial link between short-range reaction and long-range reasoning (Hexmoor & Kortenkamp, 1995).

The proposed architecture - SPOTT - addresses the problematic shortcomings<sup>24</sup> of existing behavioral architectures. A task command language lexicon based on a minimal set of navigational verbs (Miller & Johnson-Laird, 1976) and a minimal spanning subset of spatial prepositions (Landau & Jackendoff, 1993) has been developed in order to specify a collection of tasks which SPOTT can perform. The tasks are executed by the interpretation and execution of a TR+ program alongside concurrent dynamic path planning execution. The TR+ programs easily permit the addition of new sensors, actuators and control strategies. SPOTT is able to guarantee task completion for certain task and environmental configurations<sup>25</sup>. The GUI module provides visualization of the TR+ control program deliberation.

<sup>24</sup>Scalability, generalizability, predictability (i.e., task completion guarantee), and the ability to monitor execution.

<sup>25</sup>SPOTT is able to guarantee task completion when the location of the goal is known and a CAD map is available a priori.

as well as a view of the robot's state with respect to its environment. SPOTT is also modular and is dynamically responsive in real-time. Intelligent responsiveness is achieved by applying *real-time AI* techniques as well as distributing the processing across an available set of computational processor resources.

The following chapters of the thesis elaborate upon the various facets of SPOTT, namely, TR+ control, path planning, and the task command lexicon used for specifying navigational tasks. Subsequent chapters discuss the implementation of SPOTT and experimental results.

## CHAPTER 3

---

### Control: Teleo-Reactive+ Programs

In control theory, it is customary to think of the control mechanisms as being composed of analog electrical circuits, which are ideal for continuous feedback. In contrast, computer science is familiar with such ideas as sequences, events, discrete actions and subroutines. These are at odds with the notion of continuous feedback. There has been little effort to import fundamental control theory ideas into computer science (Nilsson, 1992).

Nilsson (1992; 1994) has proposed a formalism, called “*circuit semantics*”, which is conceptually similar to electrical control circuits. He has also proposed a programming language called *Teleo-Reactive*<sup>1</sup> (TR) programs, which continually get recompiled into *circuit semantics* at execution time. The TR formalism permits robot control to be programmed in a similar fashion to conventional programming (i.e., parameter passing and binding, hierarchy, recursion, and a main program with subroutines). A TR program is a list of hierarchically-ordered condition-action rules (i.e., like a production system). It has a graph representation - called a TR tree - where a condition is specified by a node and an action by an arc.

SPOTT uses an extension of Teleo-Reactive (TR) programs - called TR+ programs - for specifying behavioral control. During execution, the TR+ interpreter continuously activates and evaluates the TR+ programs. The execution of the programs is asynchronous and concurrent. The extensions to TR programs - TR+ programs - formalize the operators for condition and action expressions<sup>2</sup> and present the programmer with the ability to control how and when the expressions are evaluated.

---

<sup>1</sup>The actions of the robot are influenced by its goals, hence the term *teleo*. This term was coined by Nilsson in (1992).

<sup>2</sup>Conditional expressions using AND and OR operators are also possible in the TR formalism, but without the expressibility of TR+ programs. How and when the expression is evaluated can be controlled by the TR+ operators. In addition, the TR formalism does not contain any operators for specifying action expressions.

## 1. Teleo-Reactive Programs

Teleo-Reactive (TR) programs have been recently introduced by Nils Nilsson (1992; 1994) as a formalism for specifying event-driven behavioral control. The idea behind TR programs is to give the control system the expressive power of a computer program. The ideas that have been retained from computer science are as follows: (1) The programs have parameters that can be bound at run time and passed to subordinate routines; (2) There is a hierarchical organization; and (3) The programs can be recursive.

TR programs are very similar to an ordered set of production rules (i.e., condition-action pairs). They are interpreted in a manner comparable to the way a production system is interpreted: The list of rules is scanned from the top for the first rule whose condition part is satisfied, and then the corresponding action is executed. A TR program differs from a production system in that all the active<sup>3</sup> conditions are continuously evaluated and the action associated with the current highest TRUE condition is always the one executed. The actions can be energized<sup>4</sup> or ballistic<sup>5</sup>, whereas production system actions are only ballistic.

A hierarchical order on the list of condition-action rules is imposed by the designer. A TR program is constructed so that for each rule,  $K_i \rightarrow a_i$ , condition  $K_i$  is the regression, through action  $a_i$ , of some particular condition higher in the list.  $K_i$  is the weakest condition such that the execution of action  $a_i$  (under the most probable operating circumstances) achieves some particular condition  $K_j$  which is higher in the list ( $j < i$ ). The condition  $K_1$  is the goal condition the program is designed to achieve. The execution of the actions in a TR program ultimately achieves the goal. Should an action have an unexpected effect, the program will nevertheless continue working towards the goal<sup>6</sup>.

A TR program is organized like a computer program with a single *main* program and a collection of *subroutine* programs. When the term “*TR program*” is used, it actually refers to a collection of TR programs: one is the main program and the rest are subroutine programs. Initially, the execution of a TR program is such that the conditions in the main program are continuously evaluated (i.e., active). If one of the actions calls a subroutine<sup>7</sup> TR

<sup>3</sup>Not all conditions are continuously evaluated. A TR program is usually modular in organization, consisting of a main program and a collection of subroutines, similar to the organization of a computer program. Only the conditions in the main program (i.e., invoked by the user) and the conditions in the currently called set of subroutines are *active*.

<sup>4</sup>An action is *energized* when it is sustained continually as long as its triggering condition remains TRUE.

<sup>5</sup>An action is *ballistic* if it executes to completion after the logical transition of its triggering condition from FALSE to TRUE.

<sup>6</sup>The TR sequence is *complete* if and only if  $K_1 \vee \dots \vee K_i \vee \dots \vee K_m$  is a *tautology*, which is a clause containing complementary literals. A TR sequence is *universal* if it satisfies the *regression property* and is *complete*. A *universal* TR sequence will achieve its goal condition,  $K_1$ , if there are no sensing or external errors.

<sup>7</sup>A subroutine may be invoked from the main program, from another subroutine program, or from a previous instantiation of itself (i.e., recursion).

program, then all of the subroutine's conditions are also continuously evaluated in addition to the main TR program's conditions. This is how a TR program (i.e., a subroutine TR program) is dynamically activated at execution time. If a particular subroutine has not been called via an action for a specified period of time, then the continuous evaluation of its conditions ceases. This process is called "*garbage collection*"<sup>8</sup> and is synonymous with the identically named operation in a LISP programming environment. Activating and deactivating conditions permits recursive calls to different instantiations of the same subroutine (i.e., with different input variable values).

An action -  $a_i$  - either activates an actuator directly or via some intermediate representation. An action can also be another TR subroutine. A condition -  $K_i$  - is an executable process which returns a logical value (TRUE or FALSE). The value is based on sensory inputs and world models. In addition, the process can also return other (i.e., logical or non-logical) values through variables attached to the condition. Conditions are based on the K-B model proposed by Kaebbling and Rosenchein (1990). In this model, embedded agents or situated automata are "*computer systems that sense and act on their environments*". An appropriate series of actions emerge from the interaction of the control system with the environment. The K-B model separates perception and action (see figure 3.1). The perception module monitors the environment so that the agent (i.e., robot) can establish certain beliefs of the world. Conditional predicates based on these beliefs are the inspiration for the conditions  $K_i$  in TR programs.

The conditions and actions of TR programs form a control *conceptualization* of the robot within the world. This bears some resemblance to predicate calculus, although it is not as powerful because there is no theorem proving. TR programs are only an incomplete expression of declarative knowledge. The conditions in which the declarative knowledge is presented are both functional and relational in nature. They are functional in that the values of variables can be computed and passed to other conditions, actions or subroutines. They are relational in that they represent a logical-value, either TRUE or FALSE. An action is activated when its associated condition is TRUE and is the highest-level TRUE node in the TR program. For TR programs, the universe of discourse<sup>9</sup> are the objects and features stored in the internal model, as well as the feature information continually received via the sensors. The conditions and actions are the building blocks for what constitutes a TR program.

---

<sup>8</sup>This phrase was also used by Nilsson (1992; 1994) in order to describe the cleaning procedure.

<sup>9</sup>The *universe of discourse* are the set of objects about which knowledge can be expressed.

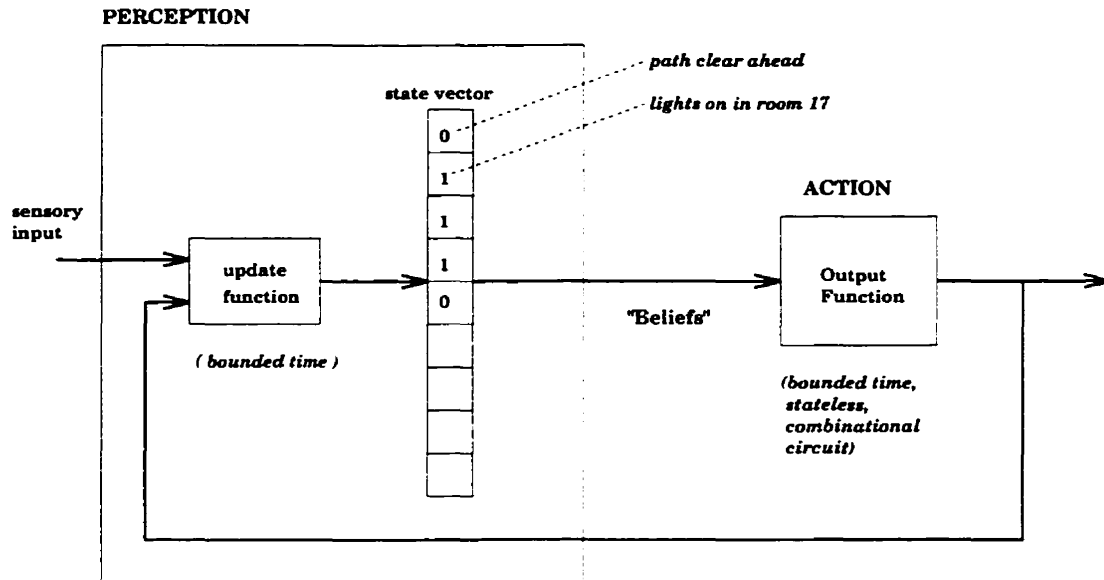


FIGURE 3.1. K-B Model of an Embedded Agent. (as adapted from (Kaelbling & Rosen-schein, 1990)) An embedded agent is a computer system that senses and acts on its environment. Beliefs of the world are continuously updated with feedback from the sensory devices.

### 1.1. TR Program Syntax

A TR program is an ordered list of condition-action rules. It is defined by a name and a list of arguments (i.e., variables) that are bound when the program is called. The binding of arguments is subject to continuous recomputation. The conditions,  $K_i$ , are Boolean combinations of components of a state vector as proposed by Kaelbling and Rosenchein (1990) (see Figure 3.1).

All the conditions in a list are evaluated concurrently. An action is determined by the highest ranking TRUE condition and is either energized or ballistic. Energized actions are the kind that must be sustained by an enabling condition to continue operating. Ballistic actions, once called, run to completion (i.e., pulse, one-shot operation). They are initiated on the logical transition of their triggering condition from FALSE to TRUE.



The syntax for a TR program as proposed by Nilsson (1992) is as follows:

```

: <TR-Prog> :: (defseq <name> <arg-list>
  : ((< KN > nil)
  : ...
  : (< Ki >< actioni >)
  : ...
  : (< K1 >< action1 >))
  : )
: <action> :: <energized-action> | <ballistic-action> | nil
  : <energized-action> :: <primitive-action> | <TR-Prog>
  : <ballistic-action> :: <primitive-action>
: <K> :: <primitive-condition> | TRUE

```

The “*primitive-conditions*” and “*primitive-actions*” are predefined by the programmer. The “*primitive-conditions*” produce a logical decision based on sensory or world model input. The “*primitive-actions*” either drive the actuator or update the world model.

## 1.2. Graph Representation

The ordered set of rules which make up a TR program can be represented in graph form, where a node represents a condition and an arc represents an action. The graph form of a TR program is called a TR tree. TR programs and its graph representation - TR trees - resemble the search trees<sup>10</sup> constructed by planning systems that work backwards from a goal condition (Nilsson, 1994). A particular type of TR tree is a TR sequence. For the general case (i.e., TR tree (see figure 3.2)), many arcs can enter a node but only one arc exits a node. A TR sequence (see Figure 3.3) is a constrained TR tree where only one arc may enter a node. TR trees (i.e., the general case) are a more appropriate representation than TR sequences when there are many actions which contribute to achieving a given condition.

Recall that all the conditions in a TR program are continuously reevaluated and the arc leaving the highest level TRUE node is executed. When represented graphically as a TR sequence, the highest level TRUE condition (i.e., node) is unambiguous (i.e., there is

<sup>10</sup> *Triangle tables* (Fikes *et al.*, 1972) are an example. They are a degenerate form of TR trees, consisting of a single path (Nilsson, 1994).

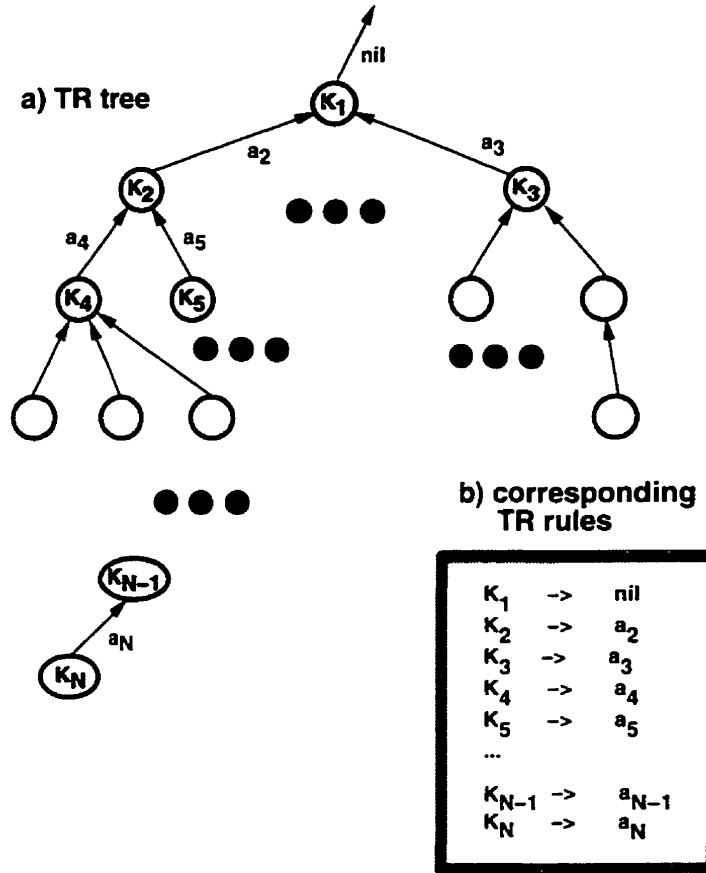


FIGURE 3.2. A TR Tree. The TR tree is shown in (a), while the corresponding list of rules is presented in (b).  $K_1$  is the program goal. The hierarchical ordering is top-down and from left to right. The nodes are numbered with respect to their ordering of importance.

only one condition per level). This is not the case when the TR program is represented as a TR tree, where many TRUE nodes can exist at the same level. Depth ties are resolved according to some prespecified ordering, usually from left to right: if many nodes are TRUE at the same level, the leftmost TRUE node's actions are triggered. The depth tie can also be resolved by finding the node, from all the TRUE nodes at the same level, with the minimum cost path. A cost is associated with each action, and a total cost is determined by finding the expenditure during the course of traversing all actions leading to the program goal node from the node in question. The highest ranking TRUE node is the one with the minimum cost path.

A TR tree can also be represented as an equivalent TR sequence. The cost for such a transformation is the loss of ability to express the goal achievement<sup>11</sup> associated with executing an action that is one of many actions which exit from the same node. The gain

<sup>11</sup>The execution of an action over time will result in a new condition being TRUE. Recall that the current TRUE condition is the regression of some higher level condition through its action (i.e., in a condition-action rule).

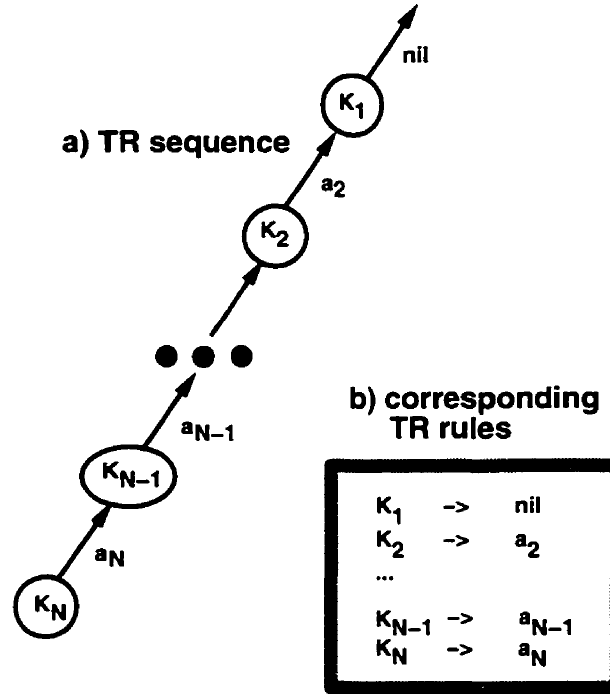


FIGURE 3.3. A TR Sequence. The graph format is presented in (a), while the corresponding list of rules is illustrated in (b).  $K_1$  is the program goal.

is that the highest level TRUE condition (i.e., node) is unambiguous because there is only one condition per hierarchical level. The TR tree in Figure 3.2 is shown in a TR sequence format in Figure 3.4.

### 1.3. TR Program Interpretation and Execution

The algorithm for the run-time interpretation and execution of a TR program is stated as follows:

- (i) *The main TR program's conditions are executed and initialized.*
- (ii) *The highest level TRUE condition specifies which  $a_i$  is activated. If  $a_i$  is a primitive action, then it is active till overridden. If  $a_i$  is a subroutine TR program, then its conditions are executed and initialized (if they have not been as of yet), and (ii) is repeated for this particular subroutine TR program.*
- (iii) *If a subroutine TR program has not been called for a predefined period of time, the execution of its conditions is stopped.*
- (iv) *If the highest level condition in the main TR program is TRUE, then the interpreter STOPS, otherwise execution returns to step (ii), using the main TR program.*

This causality for certain TR programs is explicitly shown in a TR tree (see Figure 3.2), whereas it is not in a TR sequence (see Figure 3.4).

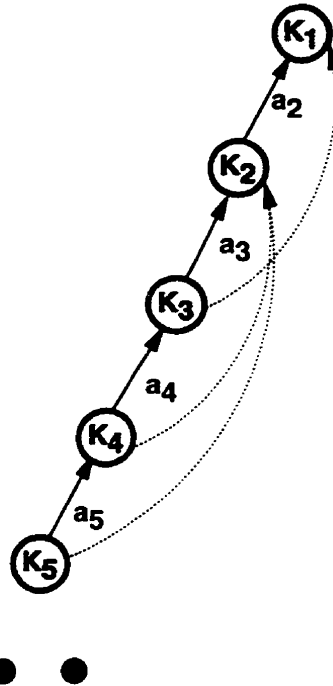


FIGURE 3.4. A TR Sequence Created From a TR Tree. With a fixed tie-breaking method, the TR tree maps to a TR sequence. This is the TR sequence which is equivalent to the TR Tree in Figure 3.2. The head of a dotted arc indicates the implicit goal achieved as a consequence of executing the action (shown by a bold arc) associated with the node. The TR Tree is a more appropriate representation because it captures this causality.

An example of the execution of the algorithm can be shown using Figure 3.5. Conditions  $c1$ ,  $c2$ , and  $c3$  in the main program are continuously evaluated. If  $c1$  is TRUE (i.e., the highest level TRUE condition in the main program), the interpreter stops because the program has satisfied its global goal. While the program is executing, the conditions in the main program (i.e.,  $c1$ ,  $c2$ , and  $c3$ ) are continuously evaluated. If  $c3$  is the highest level TRUE condition in the main program,  $a3$  is called, which invokes the subroutine  $a3$ . In this case, conditions  $c1$ ,  $c2$ , and  $c3$  as well as  $c32$  and  $c33$  are all continuously evaluated. When  $c3$  is no longer the highest level TRUE condition,  $c32$  and  $c33$  are no longer continuously evaluated.

The interpreter will only continuously evaluate the conditions that need to be evaluated. This is pertinent for writing compact code, especially when the control can be expressed as a recursive function (see Figure 3.8). Also, the computational resources will only be computing the necessary conditions, and not ones that have no influence on the resulting control. This is the major difference between the TR formalism and the other behavioral

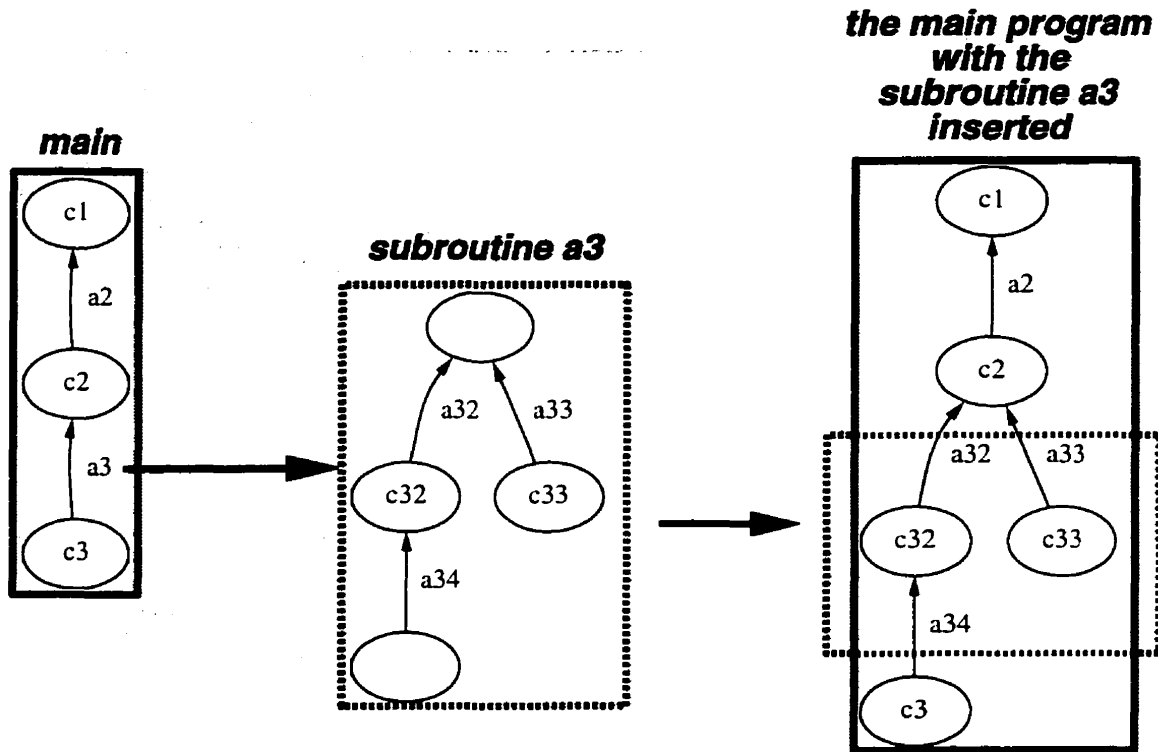


FIGURE 3.5. The Main Program and a Subroutine Program. A subroutine program has a blank condition for its head and tail conditions, as shown above. The called subroutine program is inserted into the calling main program in place of the arc which represents it (e.g., *a3*) by discarding the blank head and tail conditions. The main and subroutine programs can be equivalently represented as a single TR program as illustrated. Subroutine programs are a useful organization mechanism because they help control what constitutes an active condition. While the program is executing, the conditions in the main program are continuously evaluated. If *c3* in the main program is the highest level TRUE node, then the conditions in subroutine *a3* are also continuously evaluated, otherwise they are not.

approaches such as the subsumption architecture (Brooks, 1986). In the subsumption architecture, all specified conditions are always continually evaluated, whereas in a TR program, only the conditions pertinent to the current control<sup>12</sup> are evaluated.

#### 1.4. Circuit Semantics

Since the TR programs perform control, they must respond in bounded time to environmental changes. Conceptually, the interpretation and execution of TR programs is comparable to producing electrical-like circuits (i.e., called *circuit semantics*). They are an alternative way of visualizing the execution of a TR program. Figure 3.6 shows the corresponding circuit diagram for the TR Program in Figure 3.3.

<sup>12</sup>As specified by the TRUE conditions and the organization of the main and subroutine programs.

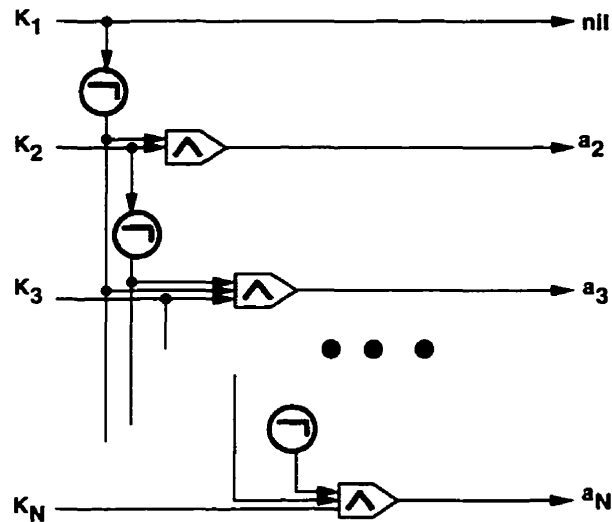


FIGURE 3.6. Implementing a TR Sequence as a Circuit: The action associated with the highest level TRUE condition is activated. (Nilsson, 1992).

Circuit semantics are similar to the finite-state machines and their composition in the subsumption architecture (Brooks, 1986). The only difference is that the TR program circuits are created at run-time, while the subsumption circuit is compiled before run-time. The *computer-program* analogy is the best approach for describing run-time circuit creation. The conditions which are evaluated depend on the calling of the subroutine TR program which contains them. If this subroutine TR program has not been called for a while, the conditions within it are not evaluated anymore. This execution style can be conceptualized by the creation and deletion of circuits. The creation of the circuits corresponds to the conditions being continuously evaluated, and their deletion implies that the conditions are no longer being evaluated on a continuous basis. In contrast, the circuits for the subsumption architecture are created before run time and execute as is. This is the major distinction between TR programs and other reactive (i.e., behavioral) programming methodologies (Brooks, 1990; Kaebbling & Rosenschein, 1990; Georgeff & Lansky, 1987; Firby, 1987; Simmons, 1994; Lyons, 1993; Simon *et al.*, 1993). All of these other methods are compiled into circuits before run time. When a particular formalism is compiled into circuitry before run-time, more circuitry is built<sup>13</sup> than needed because all possible contingencies must be anticipated. In contrast, a TR program continuously adjusts its circuitry by adding and deleting circuit elements<sup>14</sup> during execution time.

<sup>13</sup>More conditions are evaluated than required.

<sup>14</sup>Adding and deleting condition processes to and from the list of condition processes which are continuously being evaluated.

TR programs are versatile because of the blending of control theory with computer-science concepts. They can be conceptualized as a list of rules or a graph structure. The interpretation and execution of TR programs can be explained in terms of computer science concepts or as electrical-like circuits (i.e., circuit semantics).

### 1.5. TR Program Examples

TR programs are a very effective formalism for specifying event-driven control to a programmer working backwards from whatever goal condition the program is being designed to achieve. For example, figure 3.7 shows a TR program for controlling the temperature setting of a thermostat. The conditions are continuously monitored. If the temperature is low, the corresponding action is to turn up the temperature and vice versa. If the temperature is within the desired range, no action is executed.

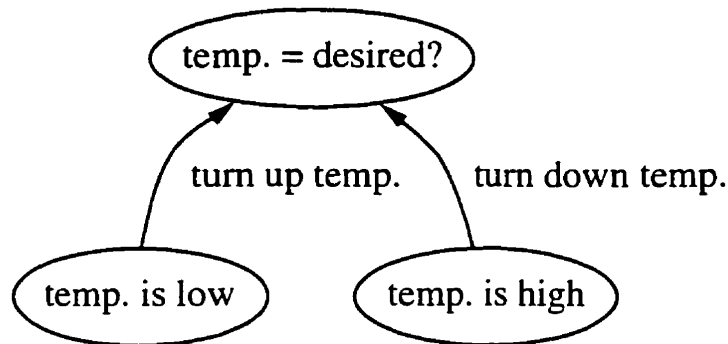


FIGURE 3.7. A TR Tree for Controlling a Thermostat. A sensor monitors the temperature and a control mechanism permits the lowering and raising of the temperature.

The “*goto(loc)*” program in Figure 3.8 illustrates a simple TR program for executing robot navigation. This program assumes that there are two actions for controlling the robot: *move*, and *rotate*. *Move* translates the robot in the direction it is currently pointing. *Rotate* turns the robot in a clockwise direction. The input to the program is a location (in  $x,y$  coordinates) for the robot to traverse to (i.e., *loc*). If the robot is not heading towards that goal position, then the robot rotates (i.e., *rotate*); otherwise, it moves towards the goal (i.e., *move*). If the specified goal position is attained, then there is no further movement.

Figure 3.9a shows how the inputs to the system - the goal specification, the position sensor and a compass reading - are combined to define the condition predicates. Figure 3.9b illustrates the circuit semantics for the “*goto(loc)*” program.

The “*amble*” program in Figure 3.8 calls “*goto*” as its subroutine. “*Amble*” recursively calls itself with intermediate goals (i.e., random positions) until there is a clear path to the

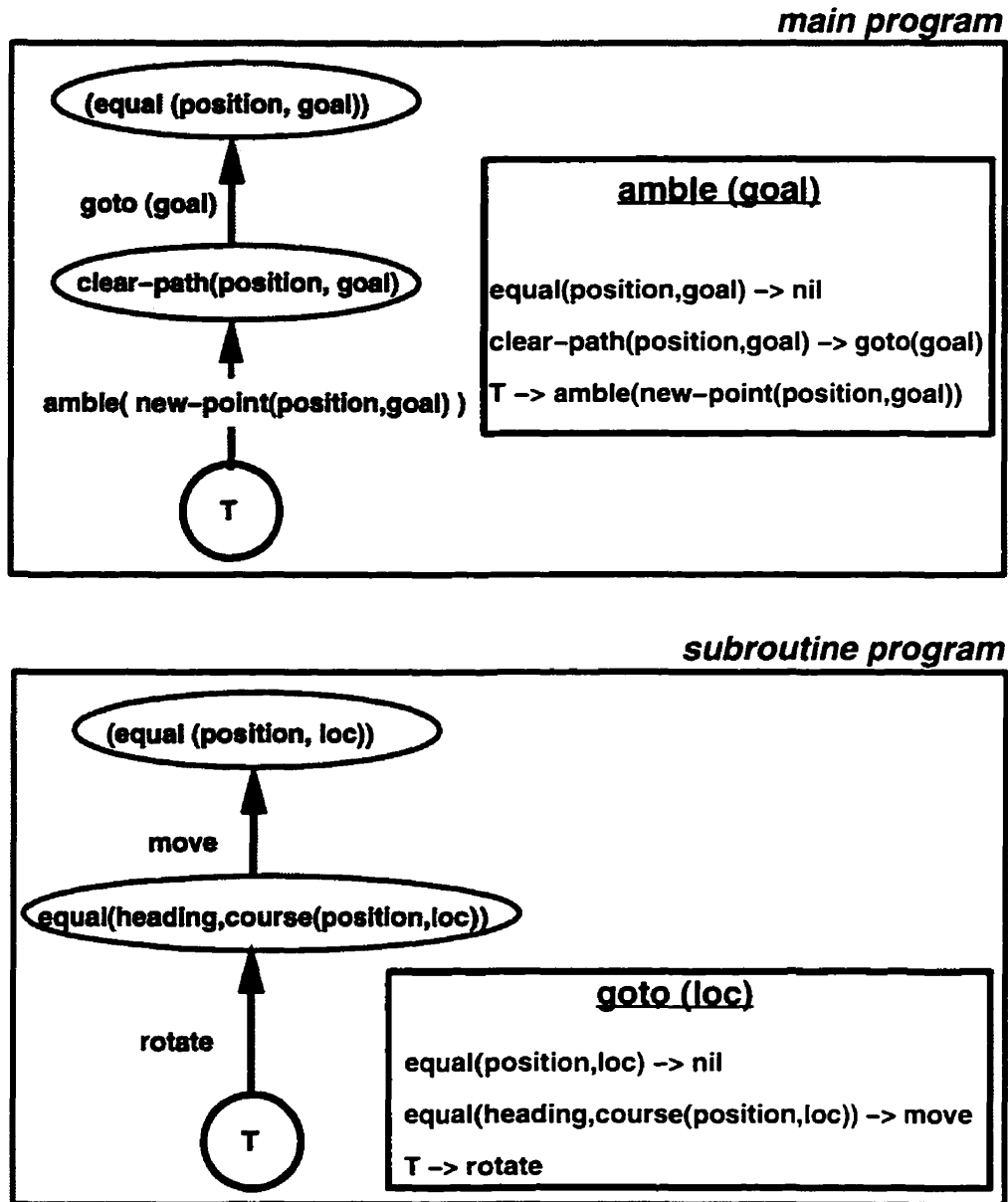


FIGURE 3.8. A Simple TR Tree for Controlling a Robot (adapted from (Nilsson, 1994)). The *goto* program causes the robot to rotate if it is not aligned with the goal, otherwise the robot moves forward. The *amble* program picks random points which serve as intermediate goals if there is not a clear path to the goal. If there is a clear path, then the *goto* program is called. This program may not be successful if appropriate random locations are not selected.

goal<sup>15</sup>. When there is a clear path to the goal (i.e., or intermediate goal), then “*goto*” is called. The undoing of all of the recursive calls results in the calling of “*goto(goal)*”, where *goal* is the original global goal (i.e., the value of *goal* when *amble* was originally called). When the recursive calls are not able to be undone, the robot is stuck in a thrashing

<sup>15</sup>This may be an intermediate goal if *amble* has called itself.



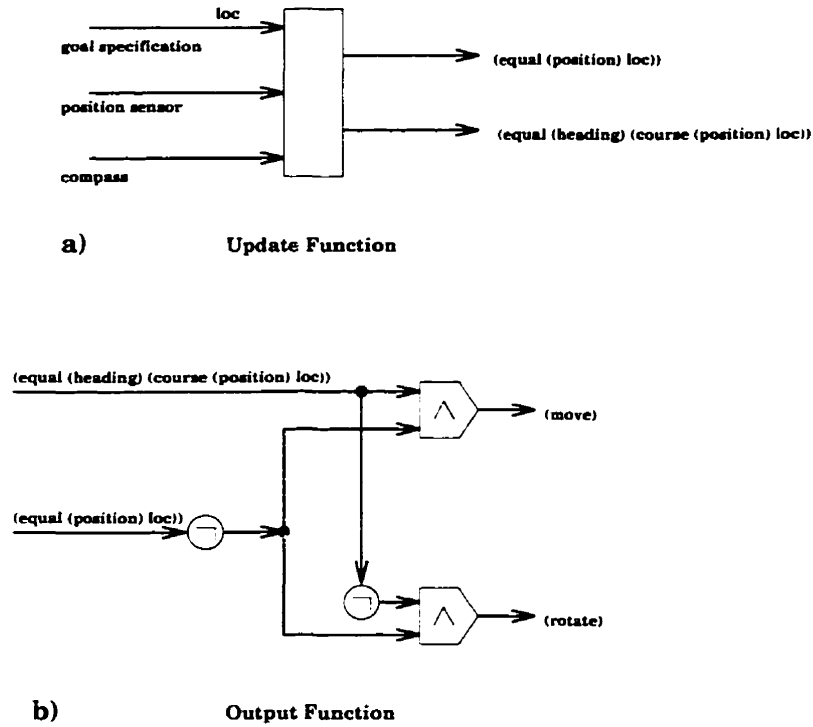
Circuitry for Moving a Robot to the Point *loc*

FIGURE 3.9. Circuit Semantics for a Simple Robot Controller: (a) This diagram illustrates how the conditional predicates in the TR program *goto(loc)* illustrated in Figure 3.8 are formed. (b) This diagram illustrates the circuit semantics for the TR program *goto(loc)* in Figure 3.8.

situation (i.e., cyclic loop) and unable to reach the global goal. There is a strong reliance on the *new-point* function to return locations that will help the robot navigate around an obstacle. When *new-point* cannot provide such locations, the robot is stuck in a cyclic loop.

The example in Figure 3.8 illustrates the TR program properties of recursion, hierarchy and variable passing. In the *amble* program (i.e., the main TR program), the lowest level condition is called recursively until the condition *clear-path* is TRUE. If there is a clear path, then the *goto* program - the subroutine TR program - is called. In both programs - *goto* and *amble* - the condition-action rules are ordered from most to least importance. The program goal is specified in the top level condition of the *amble* program: *equal(position,loc)*. Variables are used as input to conditions (e.g., *equal(heading,course(position,loc))*), and as input to a main program (e.g., *amble(loc)*) and to a subroutine program (e.g., *goto(loc)*). In this example it is possible for the robot to get stuck when no new random points (i.e., *new-point*) will put the robot in a position where it has direct sight of the goal (i.e., in order to execute *goto*).

The TR program in Figure 3.8 will work in most situations, but there is always the possibility that the robot will get stuck at a local minimum (i.e., thrashing, cyclic loop). This is a common problem for rule-based systems since all potential environmental situations are difficult to foresee as a set of hierarchically ordered set of condition-action rules. Thus, a rule-based system cannot actually guarantee that a robot will be able to circumvent all obstacles while navigating towards a goal position.

## 1.6. Advantages and Disadvantages of the TR formalism

### 1.6.1. *Advantages*

As discussed above, a TR program is goal driven (i.e., programmable goals), hierarchical, recursive, and modular. The modules are like a computer program: a main calling routine and a collection of subroutines. Variables can be passed between subroutines as well as among conditions and actions within a TR program. The biggest advantage of the TR formalism is its simple formulation and representation. A TR program is specified as a set of hierarchically ordered condition-action rules. A graphical representation - called TR trees - aids in the visualization of programming and executing TR programs.

The TR formalism is event-driven as opposed to state-driven. The programmer does not require a complete state model of the world in order to define the actions. The conditions by which an action is triggered are continuously recomputed and are responsive to sensory inputs as well as internal models. Each condition is the regression of some condition closer to the goal, through an action that nominally achieves a “*closer to the goal*” condition.

### 1.6.2. *Disadvantages*

The limitations of the TR formalism are only evident in a real-time implementation<sup>16</sup>. Two such limitations are as follows:

- (i) The execution of sensor-based conditions takes a fixed amount of time. Condition operators need to be formalized to give the programmer the flexibility to take this into account.
- (ii) Actions can either be dependent on or independent of each other. In order to specify this, it is necessary to define procedural and concurrent action operators. There are no such operators in the TR formalism.

<sup>16</sup>Nilsson's experiments with TR programs used off-line computation (i.e., a LISP machine). Simulations using a virtual world, or simulating the concurrency of executing TR programs, are not adequate for evaluating the TR paradigm.

Nilsson makes the assumption that all conditions are continuously recomputed and that they are instantaneously responsive to environmental changes. In reality, certain conditions may be computed very quickly, whereas others may require more time. The time required to evaluate a condition depends on the processing complexity. For example, acquiring sonar data requires multiple scans in order to have a dense and reliable measurement, which may take a few (i.e., two to five) seconds. There is also no way of turning off the computation of a condition within a TR program. This is useful when it is obvious that the processing of a condition is irrelevant in a particular contextual setting.

The robot should be able to perform many actions concurrently. However, a TR program has a single program goal and executes only one action at a time. The goal at the top of the main TR program is the program goal while all other nodes (i.e., conditions) in the program are intermediate goals. When a goal is expressed as a collection of conditions, the method by which the operators (i.e., AND, OR) dictate the order and concurrency of computation should be specified. Likewise, the arcs (i.e., actions) should consist of a set of actions in order to encode independent actions which may be concurrently executed.

In this thesis, an extension, called TR+ programs, is proposed to address the problematic issues of TR programs. The extension remains faithful to the basic concepts of TR programs and its graphical representation. TR+ programs permit real-time operation, concurrent actions, conditional expressions, and a level of expressiveness with conditions and actions that was not conceivable with TR programs.

There are no clear rules for choosing conditions, actions and their combinations when writing TR programs. Behavioral patterns of biological systems may provide pointers. A certain level of experience needs to be obtained from experimenting with the TR formalism before creating any TR programming rules. A start to defining a TR+ programming methodology is to experiment with a particular application (e.g., mobile robot navigation) within a specific control architecture (i.e., SPOTT). In the SPOTT control architecture, a path planner operates alongside the TR+ interpreter. This alleviates the TR+ program of the task of obstacle avoidance<sup>17</sup> and actually simplifies the taxonomy of the required functionality for mobile robot navigation. Section 2.2 provides guidelines for writing TR+ programs for mobile robot navigation within the SPOTT architecture.

---

<sup>17</sup>The TR program example in Figure 3.8 illustrates that it is difficult to guarantee successful operation of the obstacle avoidance task when encoded in a rule-based system, such as TR or TR+.

## 2. Teleo-Reactive+ Programs

Nilsson's TR formalism has been extended to handle the execution of TR programs in real-time. Simulations using a virtual world, or simulating the concurrency of the execution of TR programs is not adequate for evaluating this formalism. The real-time extension - called TR+ - is a concurrent implementation using a message passing paradigm across a network of computers.

Message passing software is used to distribute the computation of the conditions across a collection of processors. The computational-processor resource management is handled by a software package called PVM (Parallel Virtual Machine) (Geist *et al.*, 1993). The TR+ program is interpreted by asynchronously scanning the list of active condition processes<sup>18</sup>, which are executed concurrently and independently<sup>19</sup> of the TR+ interpreter.

TR programs do not permit condition expressions and more than one action to be driven from a node. This has been remedied in the TR+ formalism. In the latter, a set of actions can be performed in parallel or in sequential order. The concurrent operator is specified by  $\parallel$ , and the sequential operator is symbolized by  $\succ$ . The concurrency operator forces the left and right side operands (i.e., actions) to be executed concurrently, while the sequential operator orders the execution of the actions from left to right<sup>20</sup>.

The necessity for providing the capability of executing actions in parallel originates from the characteristics associated with solving the mobile robot navigation problem. There are certain subproblems that are not dependent on each other and should be solved concurrently; namely, mapping, localization and the search for a goal destination. This is possible in the SPOTT architecture because the parameters associated with these subproblems are independent inputs to the local path planner. The local path planner accepts the robot position (i.e., localization), the obstacles (i.e., mapping) and goals as independent inputs from a TR+ program (see Figure 4.1). The capacity to execute different problems simultaneously is not possible in the original TR formalism. Recall that an action in a TR program can either be a primitive (i.e., drives an actuator or updates a world model) or another TR program (i.e., subroutine). Each TR program (i.e., main or subroutine) is interpreted by the parallel computation of a hierarchically ordered set of condition processes in order to

<sup>18</sup>The conditions that are part of the main TR+ program are always *active*. The conditions associated with a subroutine are also *active* if the subroutine has been called on a regular basis by the main or another subroutine program (see Section 1).

<sup>19</sup>Although the condition processes depend on the TR+ interpreter for the start and termination of their execution.

<sup>20</sup> $A \parallel B$  means that actions  $A$  and  $B$  are executed concurrently.  $A \succ B$  means that action  $A$  is executed first, followed by action  $B$ , once action  $A$  has completed execution.

determine which single primitive action (i.e., or subroutine) gets executed. The concurrent operator - in the TR+ formalism - permits more than one primitive action to be executed in parallel. This operator introduces a second level of concurrency to the original TR formalism by providing the opportunity for many action primitives to be executed simultaneously. The circuitry analogy for a simple TR+ program with a concurrency operator is illustrated in Figure 3.10. The parallel operator is the major distinction between the TR+ and TR formalisms. The sequential operator is included to provide the programmer with the flexibility of controlling the order in which actions are executed when dependency is important.

The execution time of some conditions may be slower than the time required for a single TR+ interpreter to scan through the list of active conditions. A condition process is computed at a certain frequency depending on its processing complexity. The TR formalism assumed that the logical value of a condition value was always updated at each interpreter cycle. In reality, this is not always possible. In the TR+ formalism, the logical value associated with a condition process during its computation can either be set to its last computed value or to FALSE. The latter is referred to as a *ballistic* condition, while the former is called an *energized* condition. A TR+ condition's computation frequency can also be specified by the programmer.

An expression of conditions is created by the logical operators AND and OR. Recently, Benson and Nilsson (1995) have also introduced an AND operator to create an expression of conditions in an extension to the original TR formalism. The TR+ formalism includes the TR additions proposed by Benson and Nilsson (1995) as well as specifies AND ( $\wedge$ ) and OR ( $\vee$ ) based operators which permit the programmer to specify how and when the expression is actually evaluated (i.e., see Tables 3.1 and 3.2 for the definitions of the TR+ condition operators:  $\bar{\wedge}$ ,  $\bar{\wedge}_t$ ,  $\bar{\vee}$ ,  $\bar{\vee}_t$ ,  $\wedge$ ,  $\wedge_t$ ,  $\vee$ , and  $\vee_t$ ). The condition expression is either computed from left to right (i.e., by using one of the  $\bar{\wedge}$ ,  $\bar{\wedge}_t$ ,  $\bar{\vee}$  or  $\bar{\vee}_t$  operators) or all condition processes are evaluated in parallel (i.e., by using one of the  $\wedge$ ,  $\wedge_t$ ,  $\vee$ , or  $\vee_t$  operators). If the expression is evaluated from left to right, then when its logical value is known, no further conditions are processed in the expression. For example, consider the expression  $A\bar{\wedge}B$ , which means perform  $A\wedge B$  and compute the expression from left to right. If  $A$  is false,  $B$  is not evaluated. If  $A$  is true,  $B$  is evaluated. This provides the capability of turning off the computation of certain condition processes subject to event states, and also saves computation time. If the expression is rewritten as  $A\wedge B$ , then  $A$  and  $B$  are evaluated concurrently, and once their logical values are computed, the expression's value is determined by the AND operation. The  $\wedge_t$  operator is used to specify a timeout on the evaluation of the expression. If the

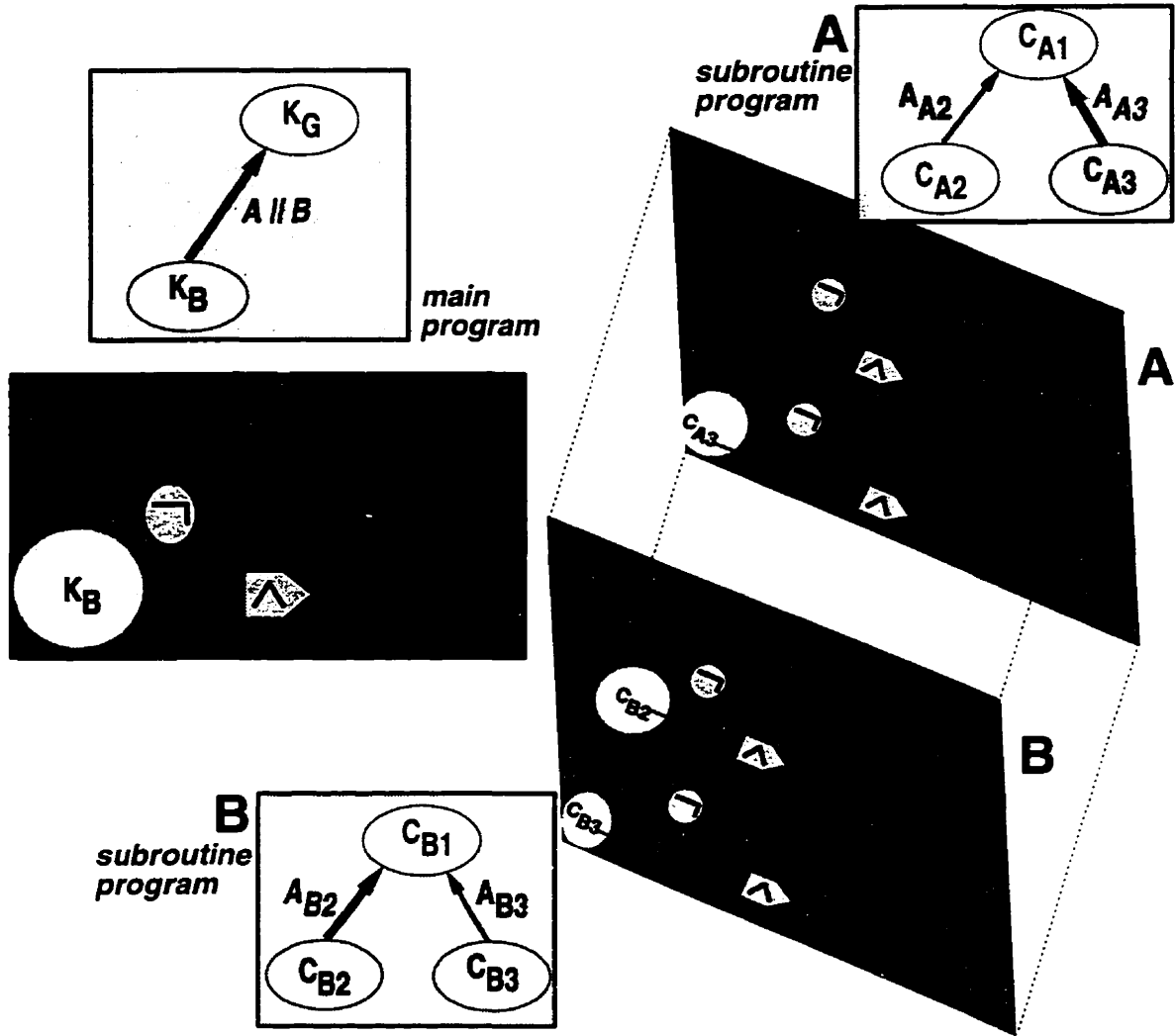


FIGURE 3.10. A TR+ Parallel Action and the Associated Circuit Semantics. In the main TR+ program, TR+ subroutines A and B are executed concurrently. The execution of A and B both culminate at any particular instance in the execution of a primitive action (i.e.,  $AA3$  and  $AB2$ , respectively). Current TRUE conditions are yellow and current active actions are red. The circuit semantics analogy is used to show that all conditions are continuously evaluated, in addition to the concurrent determination of two different primitive actions.

computation of the expression and its operands takes longer than a specified timeout period, then the value of the expression is FALSE regardless of the computed logical values of the operands. The  $\bar{\wedge}$  operator combines the ordering of the computation with the timeout feature. Similar operators are available for the OR operator:  $\bar{\vee}$ ,  $\bar{\vee}_t$ ,  $\vee$ , and  $\vee_t$ . The truth tables for the binary conditional operators are specified in Tables 3.1 and 3.2.

In addition to the two binary operators, there are two unary conditional operators: the negation operator ( $\neg$ ) and the empty set operator ( $\emptyset_t$ ) (see Table 3.3). The empty set

Operator	Name	A	B	Result	Order of Computation
$\wedge$	AND	T	T	T	Concurrent
		T	F	F	Concurrent
		F	T	F	Concurrent
		F	F	F	Concurrent
$\wedge_t$	AND(elapsed)	T	T	T*	Concurrent
		T	F	F	Concurrent
		F	T	F	Concurrent
		F	F	F	Concurrent
$\bar{\wedge}$	AND(sequential)	T	T	T	A,B
		T	F	F	A
		F	NC	F	A
$\bar{\wedge}_t$	AND(sequential,elapsed)	T	NC	T*	A
		T	F	F	A
		F	NC	F	A

TABLE 3.1. Truth Table for the AND Binary Logical Operators. The operator works on an expression composed of  $A$  and  $B$ . If the allotted time has elapsed, the asterisk \* changes T (TRUE) into F (FALSE). NC means that the condition is not evaluated.

operator performs no logical function but specifies a time limit for the computation of the operand, which if not met, results in a logical FALSE being returned. If  $A$  is TRUE and the computation of  $A$  took longer than a specified time limit, then  $\emptyset_t(A)$  will be FALSE. Otherwise,  $\emptyset_t(A)$  is also TRUE. If  $A$  is FALSE, then  $\emptyset_t(A)$  is always FALSE. The negation operator changes the logical value of the operand to its opposite (e.g., from TRUE to FALSE). If  $A$  is TRUE, then  $\neg(A)$  is FALSE, and if  $A$  is FALSE, then  $\neg(A)$  is TRUE. In both examples for the unary operators (i.e.,  $\emptyset_t$  and  $\neg$ ),  $A$  can either be a condition primitive or an expression of conditions.

As with the TR formalism, TR+ programs are represented graphically as TR+ trees. The TR+ tree representation maintains the original TR tree description (Nilsson, 1992; Nilsson, 1994) because it is simpler to visualize TR+ (or TR) program execution when arcs and nodes each have a single meaning (i.e., actions and conditions, respectively). This is in contrast to the extension proposed by Benson and Nilsson (1995) for graphically representing an AND operation. In Nilsson's extensions, the TR tree is changed such that all operands

Operator	Name	A	B	Result	Order of Computation
$\vee$	OR	T	T	T	Concurrent
		T	F	T	Concurrent
		F	T	T	Concurrent
		F	F	F	Concurrent
$\vee_t$	OR(elapsed)	T	T	T*	Concurrent
		T	F	T*	Concurrent
		F	T	T*	Concurrent
		F	F	F	Concurrent
$\bar{\vee}$	OR(sequential)	T	NC	T	A
		F	T	T	A,B
		F	F	F	A,B
$\bar{\vee}_t$	OR(sequential,elapsed)	T	NC	T*	A
		F	T	T*	A,B
		F	F	F	A,B

TABLE 3.2. Truth Table for the OR Binary Logical Operators. The operator works on an expression composed of  $A$  and  $B$ . If the allotted time has elapsed, the asterisk \* changes T (TRUE) into F (FALSE). NC means that the condition is not evaluated.

Operator	Name	A	Result
$\neg$	NOT	T	F
		F	T
$\emptyset_t$	NULL(elapsed)	T	T*
		F	F

TABLE 3.3. Truth Table for the Unary Logical Operators. The operator works on the expression (or primitive) defined by  $A$ . If the allotted time has elapsed, the asterisk \* changes T (TRUE) into F (FALSE). NC means that the action is not executed.

which contribute to a particular condition are indicated by an additional arc to represent the AND operation (see Figure 3.11a). In the TR+ tree, the AND or OR conditional expression is fully expressed within a single node, but no arcs are sacrificed for explicitly displaying the regression of the AND operands (see Figure 3.11b). It is assumed that the condition which led to the left entering arc corresponds to the regression of the first operand in the



conditional expression, and all subsequent arcs are the regressions of the corresponding conditions specified by the ordering of the operands within the conditional expression.

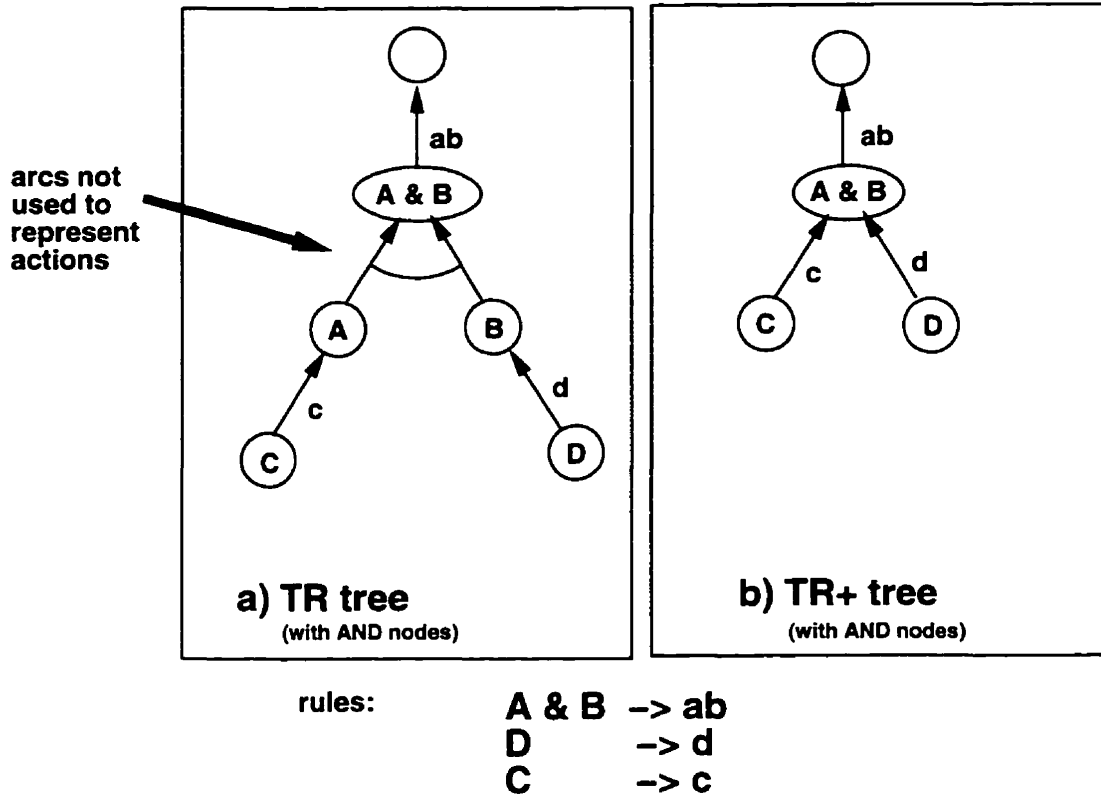


FIGURE 3.11. TR extended vs. TR+ Trees for ANDing Conditions. (a) In the extended TR tree formalism proposed by Benson and Nilsson (1995), arcs are sacrificed for explicitly representing the regression conditions for A & B. (b) This is implicit for the TR+ tree.

The TR+ formalism defines a list of AND and OR condition operators that explicitly encode the control of how the expression is evaluated. This is necessary for any real-time scenario because it gives the TR+ program designer the flexibility of specifying how and when the expressions are evaluated. This is in contrast to leaving it up to the discretion of the interpreter, as is the case in the TR formalism. Furthermore, the TR+ formalism permits concurrent primitive action execution which was not possible in the TR formalism. Independent actions should be permitted to be executed in parallel. The formalization of the operators used in a TR+ program gives the program designer the flexibility for specifying precisely the behavior of the control program in a parallel implementation.

## 2.1. TR+ Syntax

The syntax for a TR+ program based on the extensions to the TR program is expressed as follows:

```

: <TR-Prog> :: (defseq <name> <arg-list>
  : ((< KN > nil)
  : ...
  : (< Ki > < action - expi >)
  : ...
  : (< K1 > < action - exp1 >))
  : )

: <K> :: <cond-exp> | <condition> <time>
  : <cond-exp> :: <cond-u-op> <K> | <K> <cond-b-op> <K>
    : <cond-u-op> ::  $\neg$  |  $\emptyset_t$ 
    : <cond-b-op> ::  $\wedge | \vee | \vec{\wedge} | \vec{\vee} | \wedge_t | \vee_t | \vec{\wedge}_t | \vec{\vee}_t$ 
  : <condition> :: <energized-cond> | <ballistic-cond>
    : <energized-cond> :: <condition-primitive>
    : <ballistic-cond> :: <condition-primitive>

: <action-exp> :: <action> <action-op> <action>
  : <action-op> ::  $\parallel$  |  $\rangle$ 
  : <action> :: <energized-action> | <ballistic-action> |
    <action-exp>
    : <energized-action> :: <action-primitive> | <TR-Prog>
    : <ballistic-action> :: <action-primitive>

```

The *<time>* parameter associated with a *<condition>* in the definition for *<K>* is the maximum allotted computation time for a condition before it is set to FALSE by default. Similar to TR programs, the details of the specific condition and action primitives are defined by the programmer.

## 2.2. TR+ Programs for Robot Control

TR+ programs maintain the essential properties of TR programs. The extensions permit greater flexibility in programming systems for real-time operation with a dynamic and unpredictable environment. TR+ programs for navigation control within SPOTT are simple because path planning and execution are not their responsibility. The role of the TR+ program is reduced to the role of continuously providing the necessary inputs to the local path planner, namely: the current obstacle configuration (i.e., mapping), the current estimate of the robot's position (i.e., robot localization), and a goal specification (i.e., destination). A simple TR+ program which facilitates concurrent dynamic map creation and localization of the robot, while striving to meet the task objective is shown in Figure 3.12. The map creation (*new\_object\_positions\_are(\_)*) and localization (*current\_robot\_position\_is(\_)*) condition processes are based on sonar range data and were developed at CIM (Mackenzie & Dudek, 1994). In this example, the program continues executing until the task target has been reached. The environment is mapped and the robot is localized, continually at a specified frequency, while awaiting task completion. It is assumed that SPOTT's local path planner is responsible for performing obstacle avoidance and moving the robot towards the goal at the same time while the TR+ program is executing. The role of the TR+ program is to provide the necessary information to the path planner so that navigation can be carried out.

A local path planner is used in conjunction with a TR+ controller because it is difficult (i.e., probably impossible) to completely express obstacle avoidance as a collection of TR+ rules for all potential situations. Many possible obstacle avoidance contextual rules could actually be encoded as TR+ behavioral rules, but there is no guarantee that all environmental contexts would be captured. To overcome the combinatorial explosion of obstacle avoidance rules, an *independent* concurrent module performs local path planning based on a potential field technique. The latter is a representation based on a discretized grid of the local map<sup>21</sup> and avoids the combinatorial pitfalls associated with encoding all possible obstacle avoidance situations in a rule-based system. A TR+ program executes by continually providing goal, obstacle, and robot position estimate information - via the map database - to the local path planner, which in turn is responsible for issuing trajectory commands to the robot.

<sup>21</sup> The local map consists of a local window into the architectural CAD map and the collection of newly sensed features.

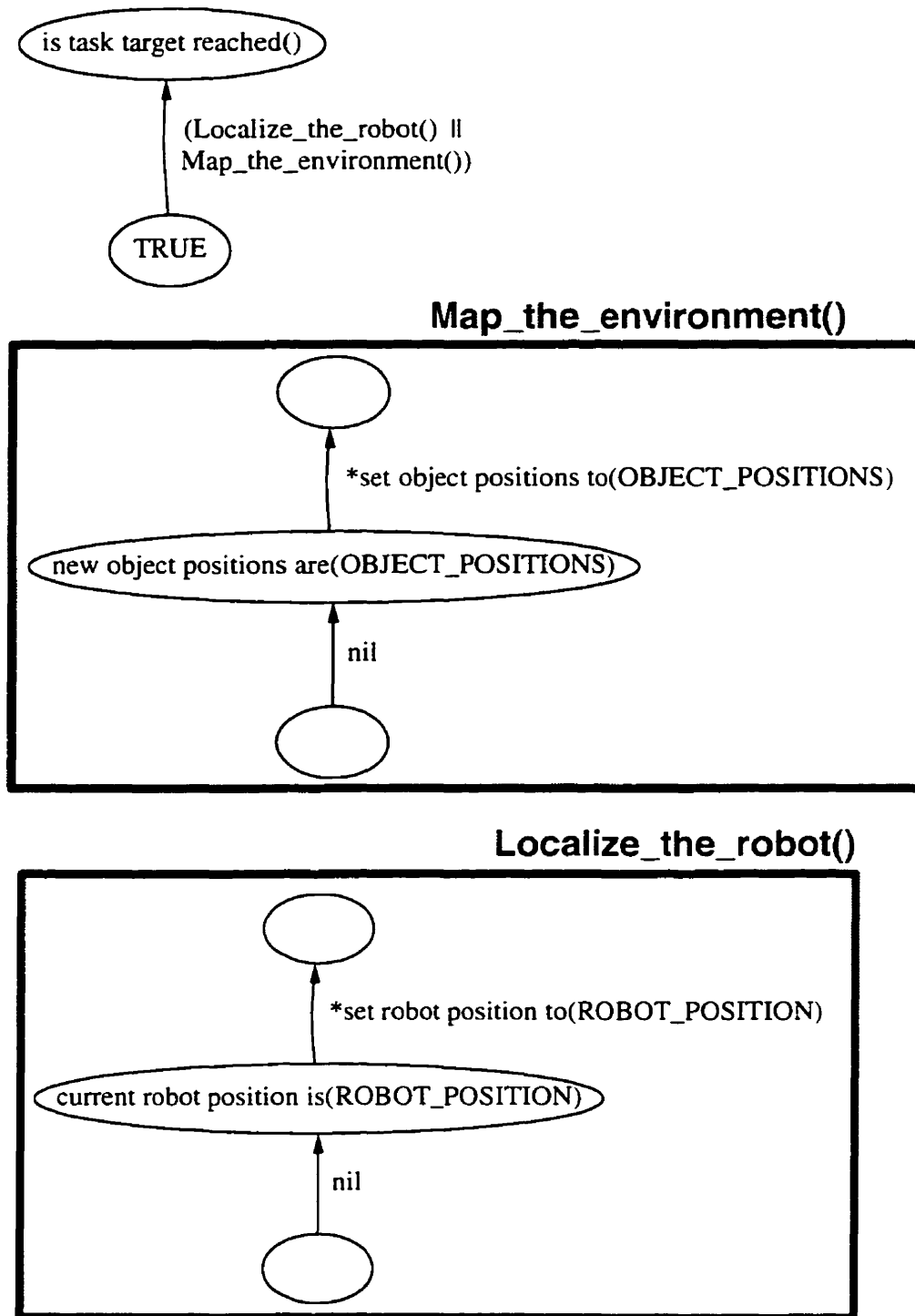


FIGURE 3.12. A Simple TR+ Program for Navigation to a Specified Location. A TR+ tree for controlling a robot navigating towards a goal position, while concurrently building a map and localizing the robot is shown. The asterisk \* in front of a condition (e.g. *set object position to* and *set robot position to*) is used to signify that the condition is ballistic.

SPOTT's organization (i.e., having a path planner function concurrently with the TR+ interpreter) helps define the role and structure of a TR+ program. SPOTT's local path planner continually requires updates of the environmental configuration (i.e., obstacles), the current estimate of the robot's position, and a goal specification (i.e., destination). The TR+ formalism permits the acquisition of this information in parallel.

The example in Figure 3.12 illustrated a simple TR+ program (consisting of a main routine and two subroutines) that performed the basic behaviors necessary for navigation control. This TR+ program assumed that the user has specified the spatial coordinates of the goal. However, if the goal's coordinates are not specified, then the TR+ program would also be responsible for performing a search within the environment by creating an ordered list of intermediate goals. In this case, the user might specify the goal as an object description, and it would be up to the perceptual capabilities of the system to identify and recognize it (i.e., using the QUADRIS sensor). The goal of the visual search is to position the robot such that the desired object is located and recognized. Additional responsibilities of the TR+ program include monitoring the homeostasis of the robot (i.e., internal failures such as battery failure, sensor failure) and assuring its safety. The latter is usually encoded as reactive behaviors (i.e., fast responses to environmental stimuli). For example, SPOTT uses the bumper and infrared sensors as a fail safe against obstacles that were not discovered while mapping the environment with the sonar and QUADRIS range data.

Ethological studies (Tinbergen, 1951; McFarland, 1989) have also proposed hierarchical control structures for modelling animal (e.g., birds, insects) behavior. The type of behaviors modelled included survival, eating, defending, attacking and mating. The listed order is usually how these behaviors were hierarchically organized. Robotics researchers have used ethology as an inspiration to formulate behaviors for mobile robots such as wall following (Connell, 1990; Gat, 1991b; Noreils & Prajoux, 1991), wandering (Anderson & Donath, 1991; Flynn & Brooks, 1988; Payton, 1986), obstacle avoidance (Watanabe *et al.*, 1992; Kweon *et al.*, 1992), position estimation (Kadonoff *et al.*, 1988), homing (Connell, 1990), and fleeing (Flynn & Brooks, 1988). Many of these behaviors can be categorized as being associated with the task of navigation<sup>22</sup>. The only exception is the fleeing behavior which is a safety reaction to a looming dangerous situation (e.g., large object approaching). In order to write an appropriate TR+ program, the two questions that the programmer must answer are: (1) What are the behaviors? and (2) How are the behaviors organized hierarchically?

---

<sup>22</sup> *Wall following* and *wandering* are a type of heuristic search strategy. *Homing* is a specification of the destination for navigation. *Position estimation* and *obstacle avoidance* are fundamentally necessary behaviors for navigation.

As discussed earlier, the type of behaviors needed include navigation, safety, and homeostasis. Navigation can be further broken down into mapping, robot position estimation and the search for the goal (i.e., if the spatial coordinates of the goal are not provided by the user). A set of behaviors may be executed concurrently when there is no dependence between them. Thus since there is no dependence between safety, homeostasis, and navigation, they may all be executed in parallel. An alternative may be to prioritize homeostasis behaviors higher than safety and navigation. This is left up to the discretion of the TR+ programmer. Also, all of the navigation behaviors (e.g., mapping, localization, search) are independent of each other, and may also be executed in parallel. Figure 3.13 illustrates the form of a typical TR+ program for mobile robot navigation. The typical TR+ program consists of a main program whose highest condition is the satisfactory completion of the task command. The main program executes a collection of subroutines whose organization is determined by the TR+ programmer, and is subject to constraints imposed by the dependencies between the behaviors. In order to improve the clarity of understanding the TR+ program, the subroutines should be organized hierarchically and modularly.

An example TR+ program<sup>23</sup> consisting of a main routine and fifteen subroutines is illustrated in Figures 3.14 to 3.22. This TR+ program is an example of mobile robot control using SPOTT's framework. For the task of navigation, writing a TR+ program is relatively simple because all of the mentioned behaviors are independent of each other and may be executed concurrently. This would not be the case if the manipulation of objects were part of SPOTT's task repertoire. In this situation, there would be a strong dependence between manipulation and navigation (i.e., the robot has to be near the object before it can manipulate it). The use of TR+ programs for providing event driven control in other fields such as telecommunications, industrial process control, or manufacturing will require the definition of the necessary behaviors as well as a determination of their inter-dependence.

---

<sup>23</sup>This TR+ program makes use of the independence of navigation, safety and homeostatic monitoring. The navigation behaviors of mapping, robot position estimation and search are also independent of each other.

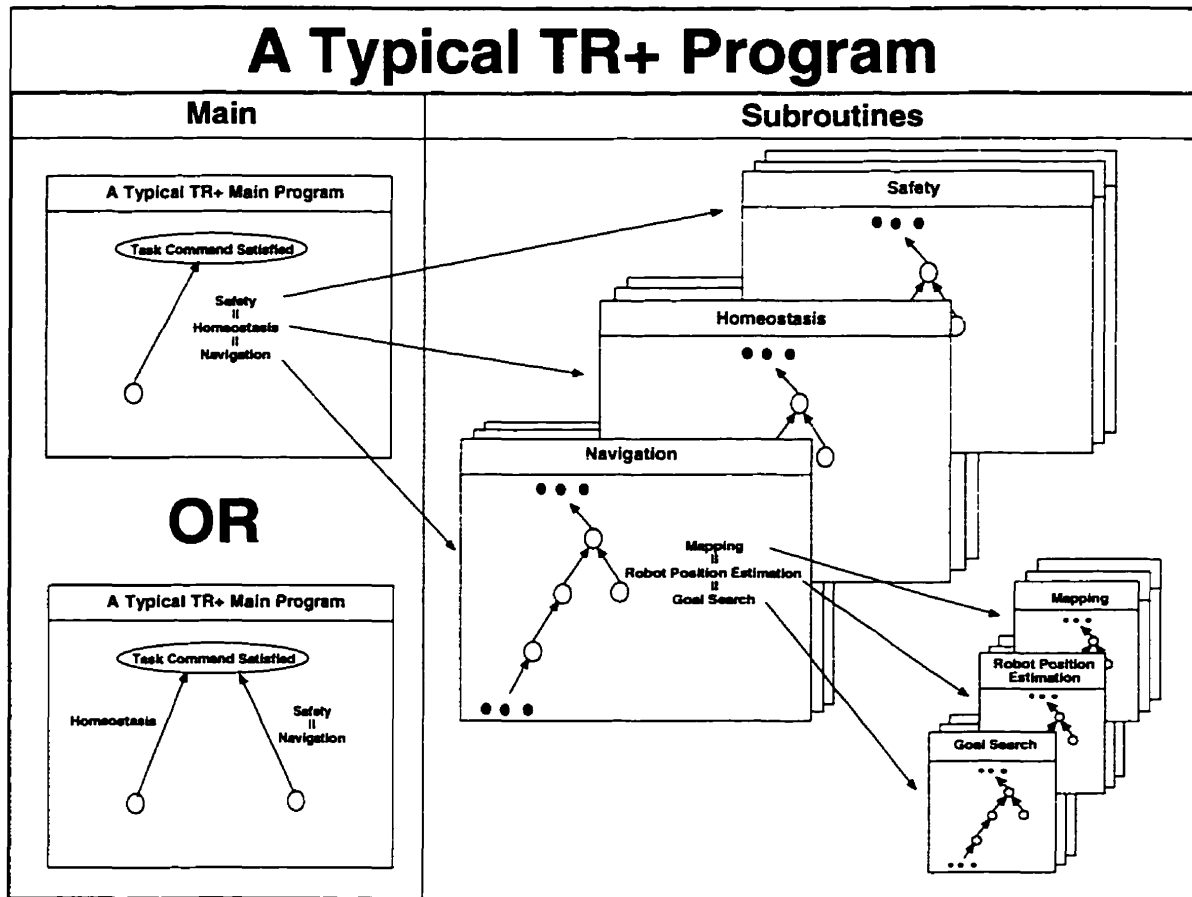


FIGURE 3.13. A Typical TR+ Program for Mobile Robot Navigation. The typical TR+ program consists of a main program whose highest condition is the satisfactory completion of the task command. The safety, homeostasis, and navigation behaviors may all be executed in parallel because they do not depend on each other. An alternative may be to prioritize homeostasis behaviors higher than safety and navigation. This is left up to the discretion of the TR+ programmer. Navigation is further broken down into a set of behaviors (e.g., mapping, localization, search) that are independent of each other, and may also be executed in parallel.

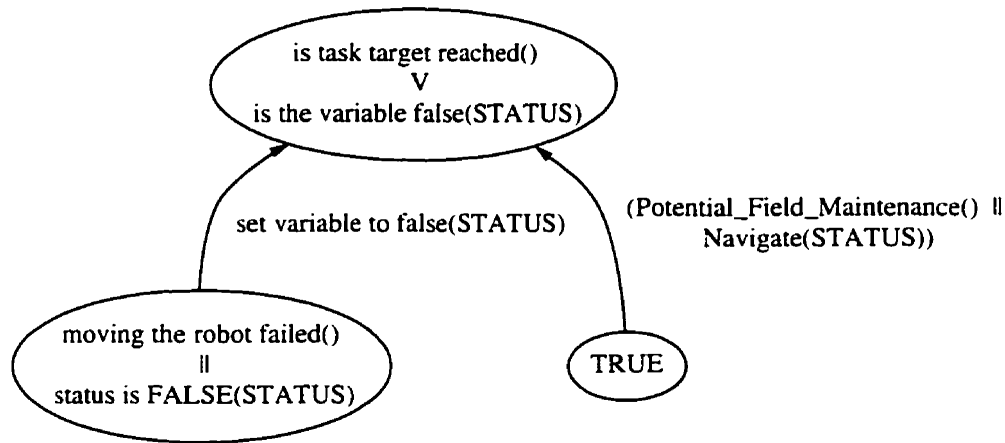


FIGURE 3.14. The Main Routine for a More Complicated TR+ Program. The *moving\_the\_robot\_failed()* condition monitors the homeostatic (e.g., battery power, loss of radio control) state of the robot. Currently it only monitors the state of the communication link. The robot will stop if the target is reached or if a homeostatic failure is annunciated. The *Potential\_Field\_Maintenance()* subroutine is shown in Figure 3.15 and the *Navigate()* subroutine in Figure 3.17.

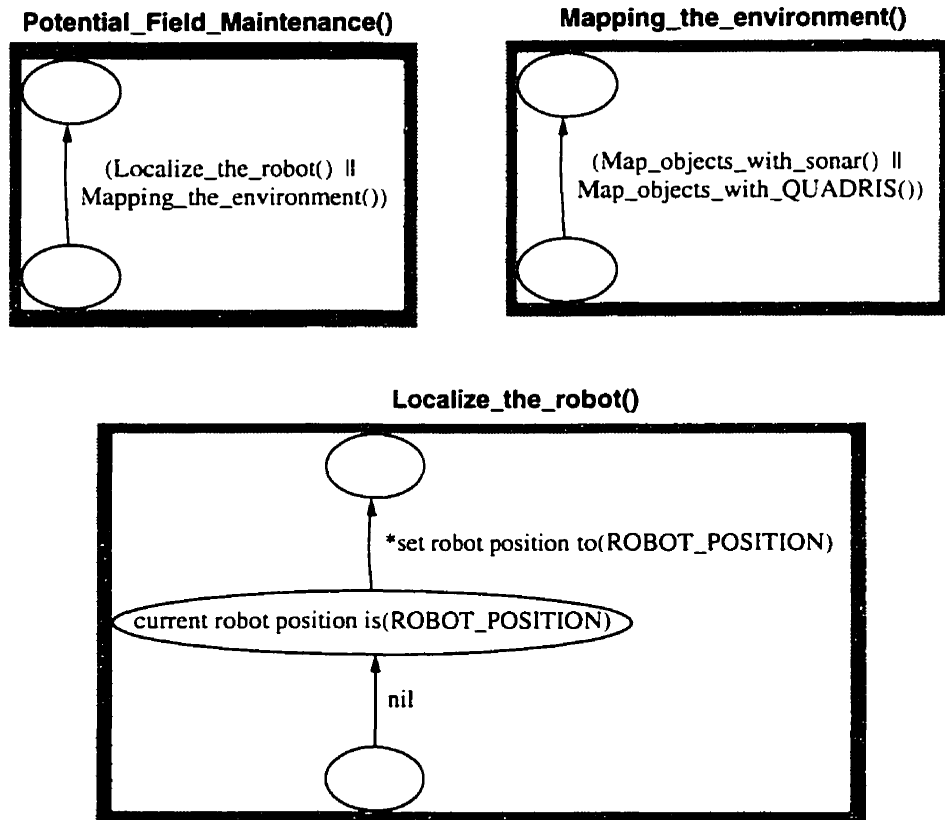


FIGURE 3.15. TR+ Subroutines Responsible for Maintaining the Potential Field. The local path planner (i.e., potential field) requires the current configuration of obstacles and the position of the robot. Concurrently, the environment is mapped (i.e., *Mapping\_the\_environment()*) and the current estimate of the robot is updated (i.e., *Localize\_the\_robot()*). *Mapping\_the\_environment()* can either be done with sonar data (i.e., *Map\_objects\_with\_sonar()*) (see Figure 3.16) or with QUADRIS range data (i.e., *Map\_objects\_with\_QUADRIS()*) (see Figure 3.16).



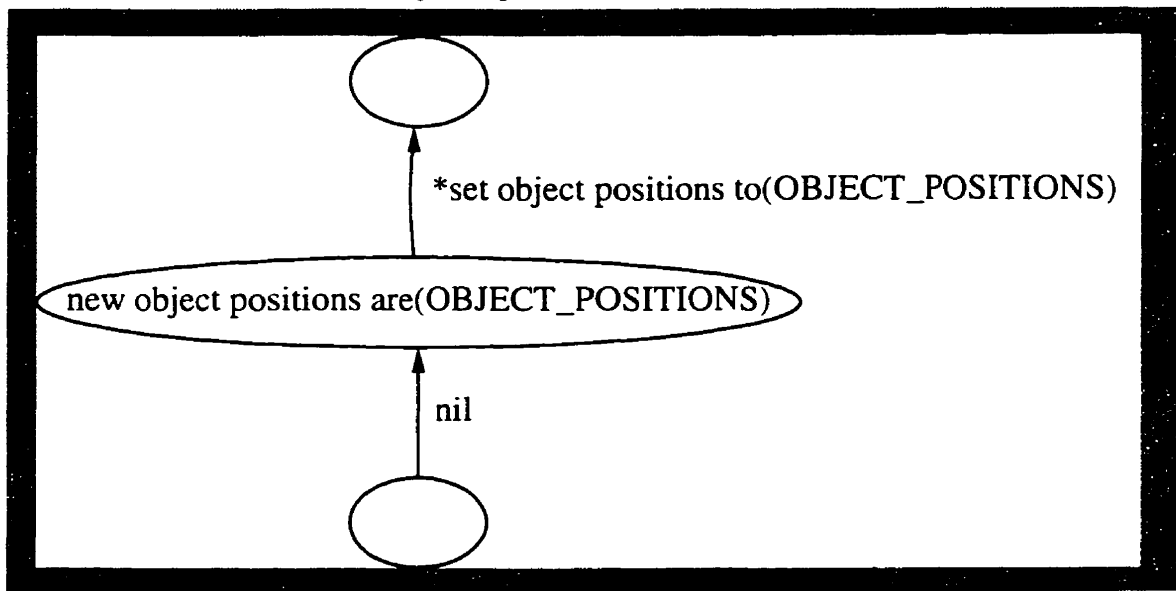
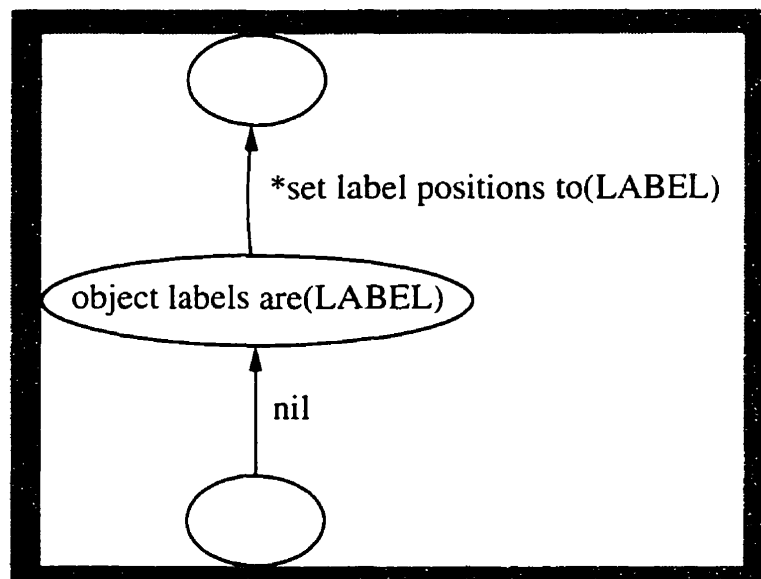
**Map\_objects\_with\_sonar()****Map\_objects\_with\_QUADRIS()**

FIGURE 3.16. **TR+ Subroutines for Mapping.** It takes between two to five seconds to collect and process a dense sonar reading. Therefore, sonar data collection is less frequent than map updates with QUADRIS data. The QUADRIS range data is collected every second (Bui, in preparation). The frequency of sonar data collection is not only based on time, but is also dependent on the distance travelled (see Chapter 6 for how distance travelled is used in conjunction with time in order to determine the frequency of collecting sonar data).

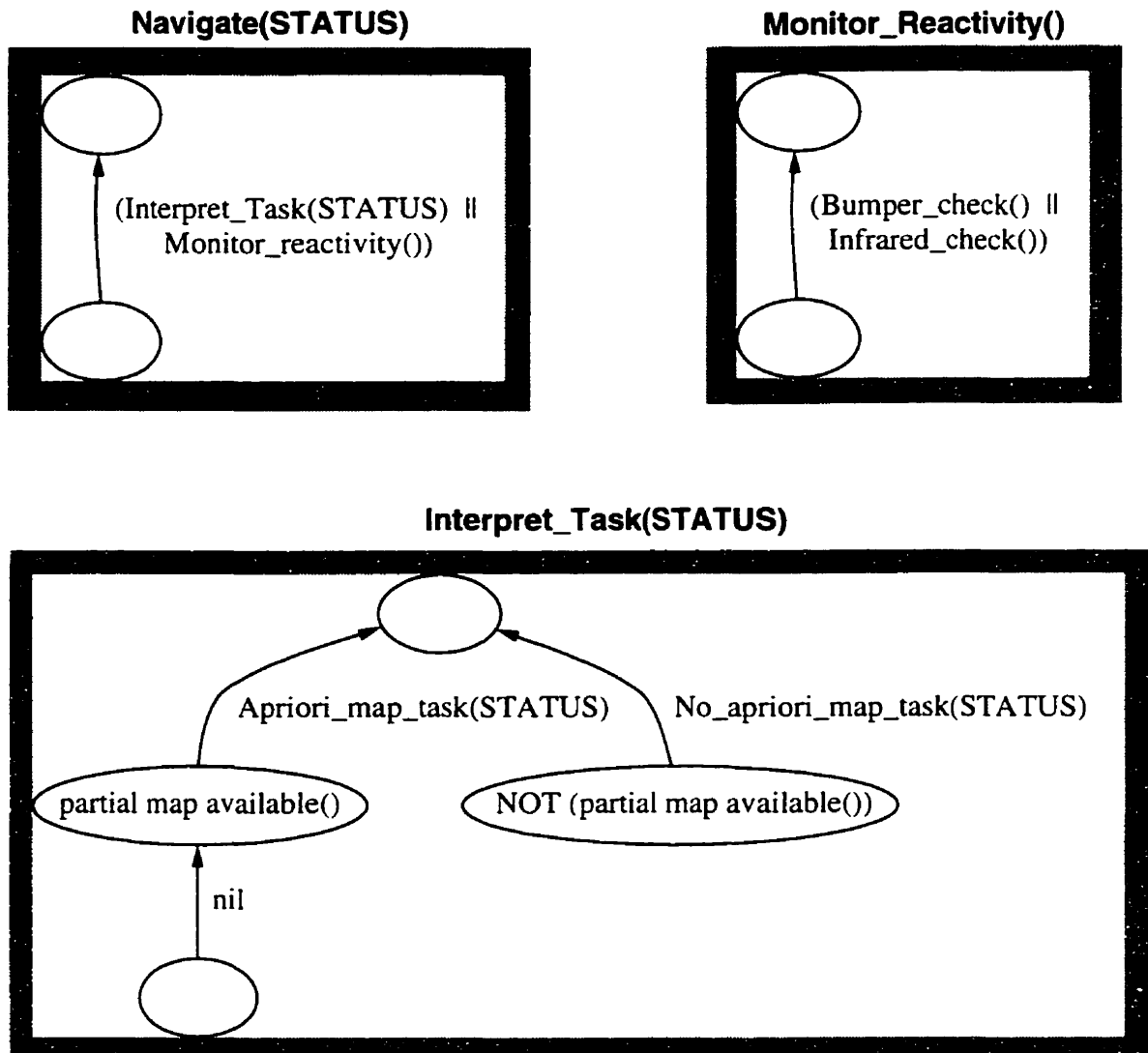


FIGURE 3.17. TR+ Subroutines for Setting the Goals for Navigation. *Navigate(STATUS)* is called from the main TR+ program (see Figure 3.14). It calls *Interpret\_Task(STATUS)* which sets the goal or search strategy. Concurrently, *Monitor\_Reactivity()* monitors the bumper and infrared data for obstacle information which was missed by the sonar or QUADRIS sensors (see Figure 3.18). *Interpret\_Task(STATUS)* will call a search strategy which is appropriate to the given a priori knowledge (i.e., CAD map is or is not available a priori). The search strategy subroutines are in Figure 3.19.

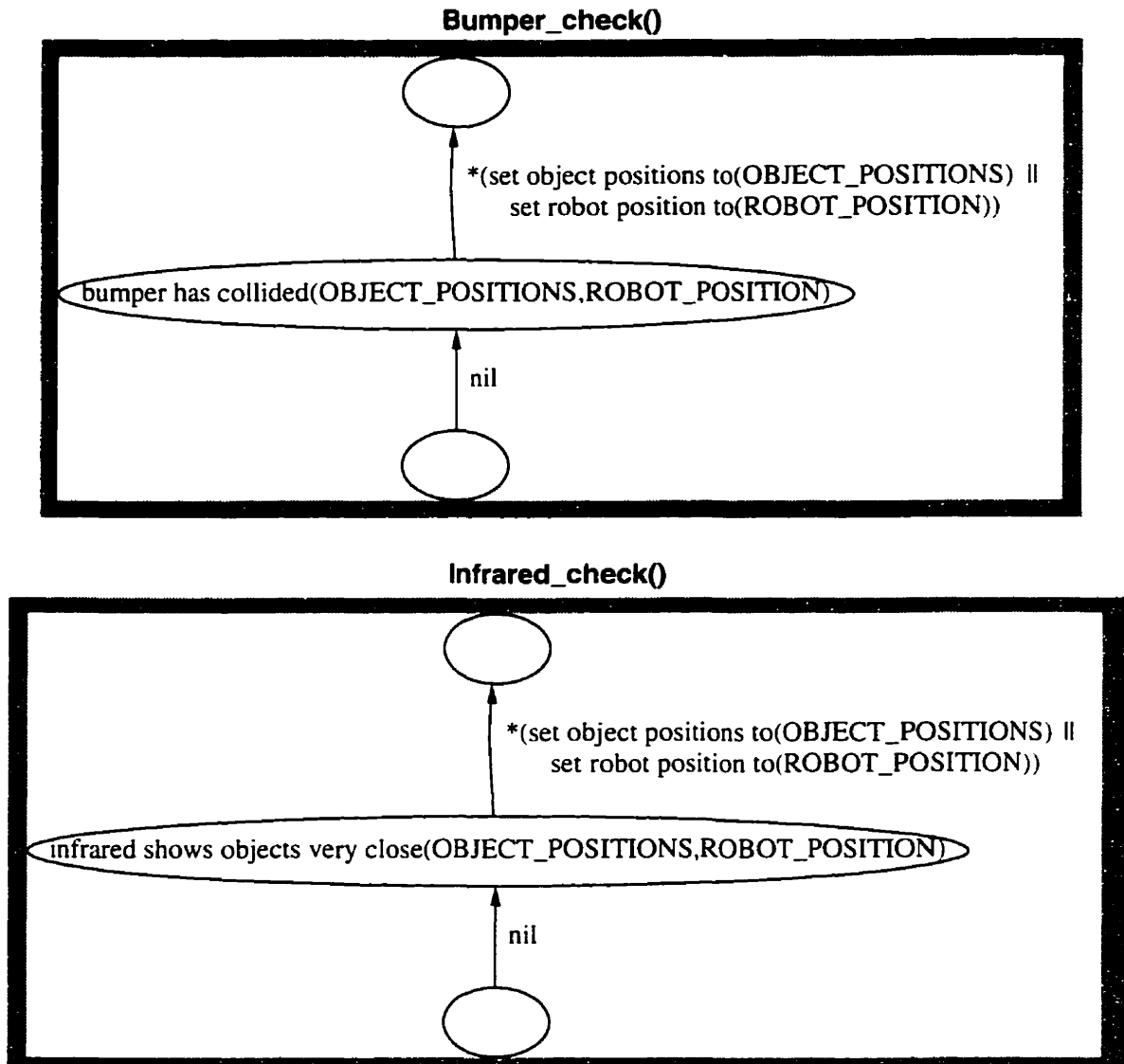


FIGURE 3.18. **TR+ Subroutines for Performing Reactivity.** *Bumper\_check()* will update the potential field of the newly discovered obstacle (*OBJECT\_POSITIONS*) and move the robot (*ROBOT\_POSITION*) slightly away from the obstacle. The potential field will be made aware of the change in robot position via the map database. *Infrared\_check()* functions in a similar fashion.

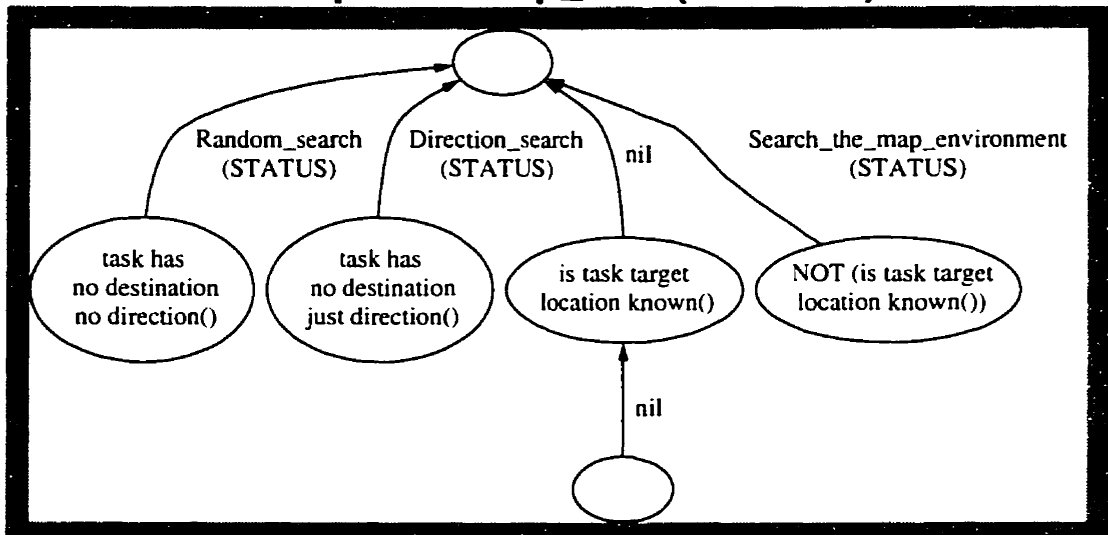
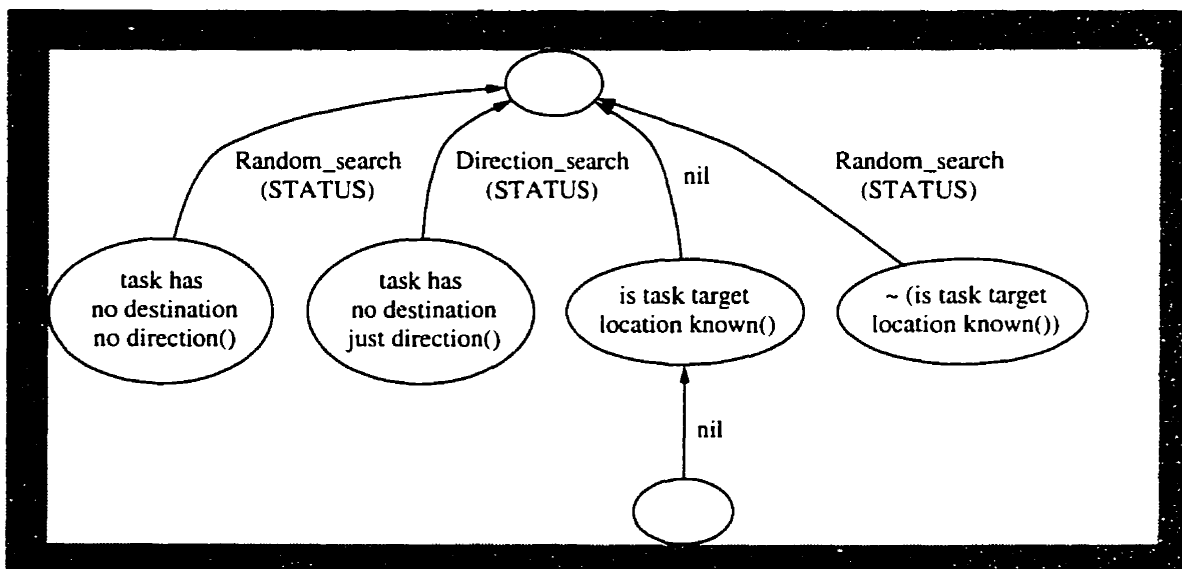
**Apriori\_map\_task(STATUS)****No\_apriori\_map\_task(STATUS)**

FIGURE 3.19. TR+ Subroutines for Deciding Search Strategies. *Apriori\_map\_task(STATUS)* is called if a CAD map is available a priori; otherwise *No\_apriori\_map\_task(STATUS)* is called (see Figure 3.17). Both of these subroutines further break down the task command specified by the user in order to determine the search strategy. The search strategy can either be random (i.e., *Random\_search(STATUS)*) (see Figure 3.22), initiated by visiting each room in the CAD map systematically (i.e., *Search\_the\_map\_environment(STATUS)*) (see Figure 3.20), or defined by a spatial preposition specifying a direction (e.g., left, right, north) (i.e., *Direction\_search(STATUS)*) (see Figure 3.21).

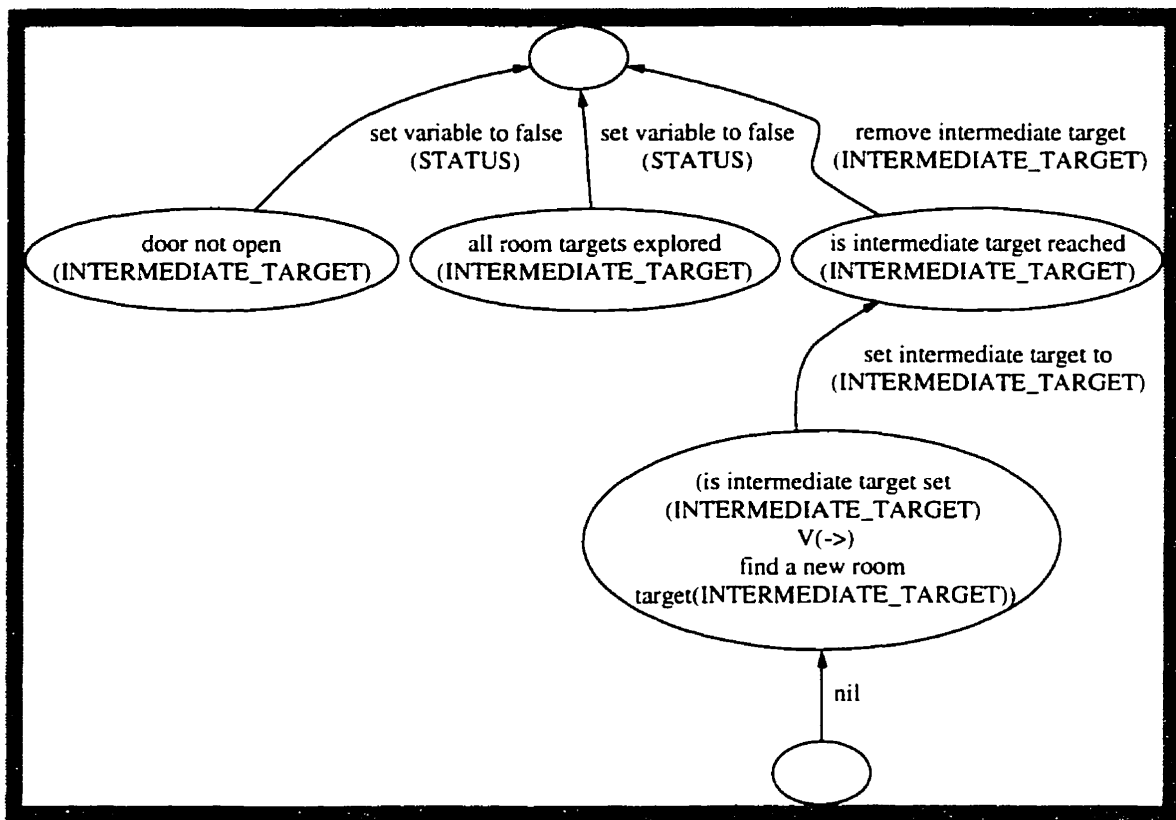
**Search\_the\_map\_environment(STATUS)**

FIGURE 3.20. TR+ Subroutines for Performing Room Search. The user specifies a description of an object to be found. The object's spatial location is unknown. The search for the object is based on setting intermediate goals specifying the rooms in the CAD map. The condition *find\_a\_new\_room\_target(.)* keeps track of the previously visited rooms and selects a new room to visit. The condition *all\_room\_targets\_explored(.)* is TRUE if all the rooms have been visited. In this case, the search fails, the STATUS variable is set to FALSE, and is propagated back to the main routine (see Figure 3.14), causing the TR+ program to stop execution. The TR+ program assumes all doors are open to start with. If a door is closed (*door\_not\_open*), the execution also stops. In this case, the user would have to open the door and restart the execution of the program. Alternatively, the subroutine can be rewritten such that the room with the closed door is ignored and returned to after all the other rooms have been searched.

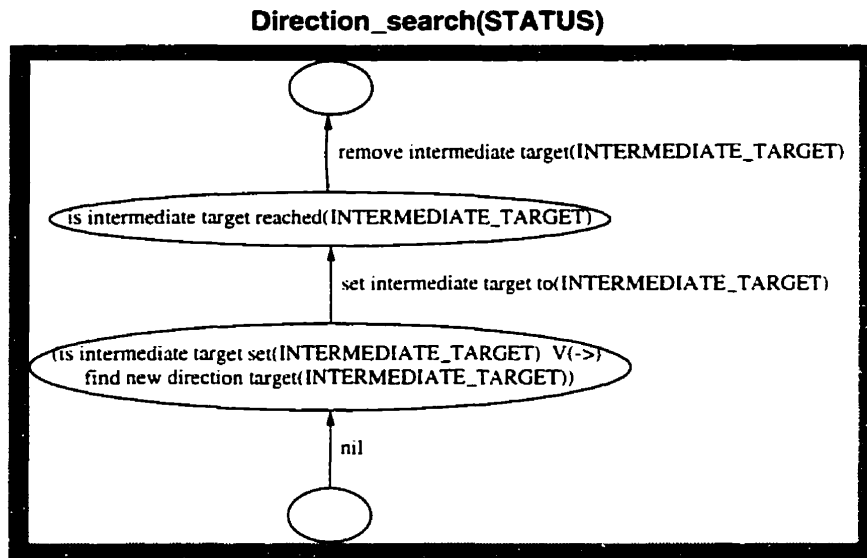


FIGURE 3.21. **TR+ Subroutines for Performing Intelligent Teleo-Operation.** An intermediate target is set based on a directional spatial preposition (e.g., left, north) which is defined by the user as part of the task command (see Figure 3.19). An intermediate target is set in the specified direction at a particular distance (e.g., 2 m) away from the current position of the robot. The *is\_task\_target\_reached()* condition in the main routine (see Figure 3.14) will be TRUE if there is an obstacle that intersects the line formed by joining the robot position with the position of the intermediate goal (i.e., if a direction is specified as part of the task command). The robot will move in the desired direction until an obstacle is encountered.

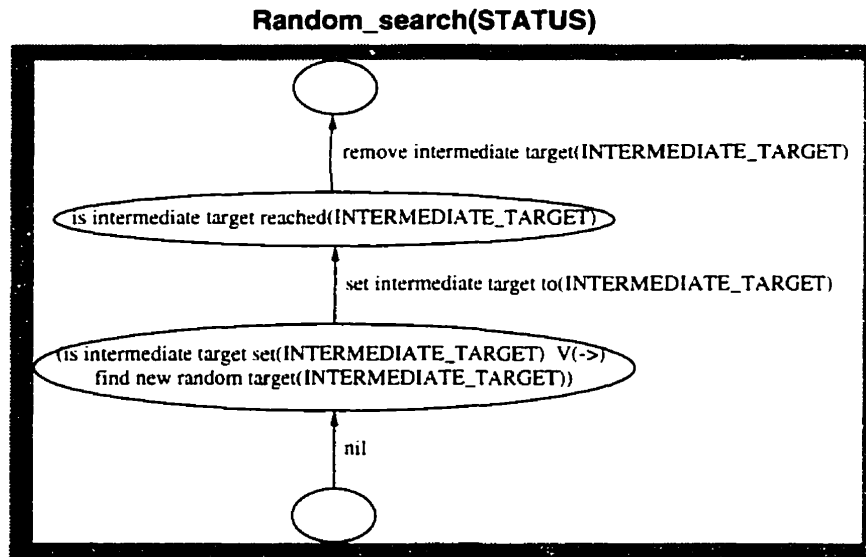


FIGURE 3.22. **TR+ Subroutines for Performing Random Search.** This subroutine performs a random search of the environment by setting destinations that are generated by a random number generator. There is no guarantee that the environment will be fully explored. This subroutine is called when there is no CAD map available a priori (see Figure 3.19).

## CHAPTER 4

---

### Path Planning

Path planning is not done using the TR+ controller because it is difficult to completely express obstacle avoidance as a collection of TR+ rules for all potential situations. Many possible obstacle avoidance contextual rules could actually be encoded as TR+ behavioral rules, but there is no guarantee that all environmental contexts would be captured. To overcome the combinatorial explosion of obstacle avoidance rules, SPOTT uses an *independent* module which performs local path planning concurrently with the execution of a TR+ program (see Figure 4.1).

Path planning is the guidance of an agent - a robot - from a source to a destination, while avoiding all encountered obstacles. A robot must not only be able to create and execute plans, but must be willing to interrupt or abandon a plan when circumstances demand it (Georgeff & Lansky, 1987). In traditional AI planning, a smart planning phase constructs a plan which is carried out in a mechanical fashion by a dumb executive phase. The use of plans regularly involves rearrangement, interpolation, disambiguation, and substitution of map information. Real situations are characteristically complex, uncertain, and immediate (Agre & Chapman, 1987). These situations require that the planning and the execution phases - of at least one of the planners - function in parallel, as opposed to the serial ordering found in traditional AI planning.

Path planning can be categorized as either being *static* or *dynamic*, depending on the mode of available information (Hwang & Ahuja, 1992). A *static* path planning strategy is used when all the information about the obstacles is known a priori. Most path planning methods are static. The path planning is *dynamic* when no or partial information is available about the obstacles a priori. The environment can also be unpredictable and time-varying. Dynamic path planning is problematic because the path needs to be continually recomputed as new information is discovered.

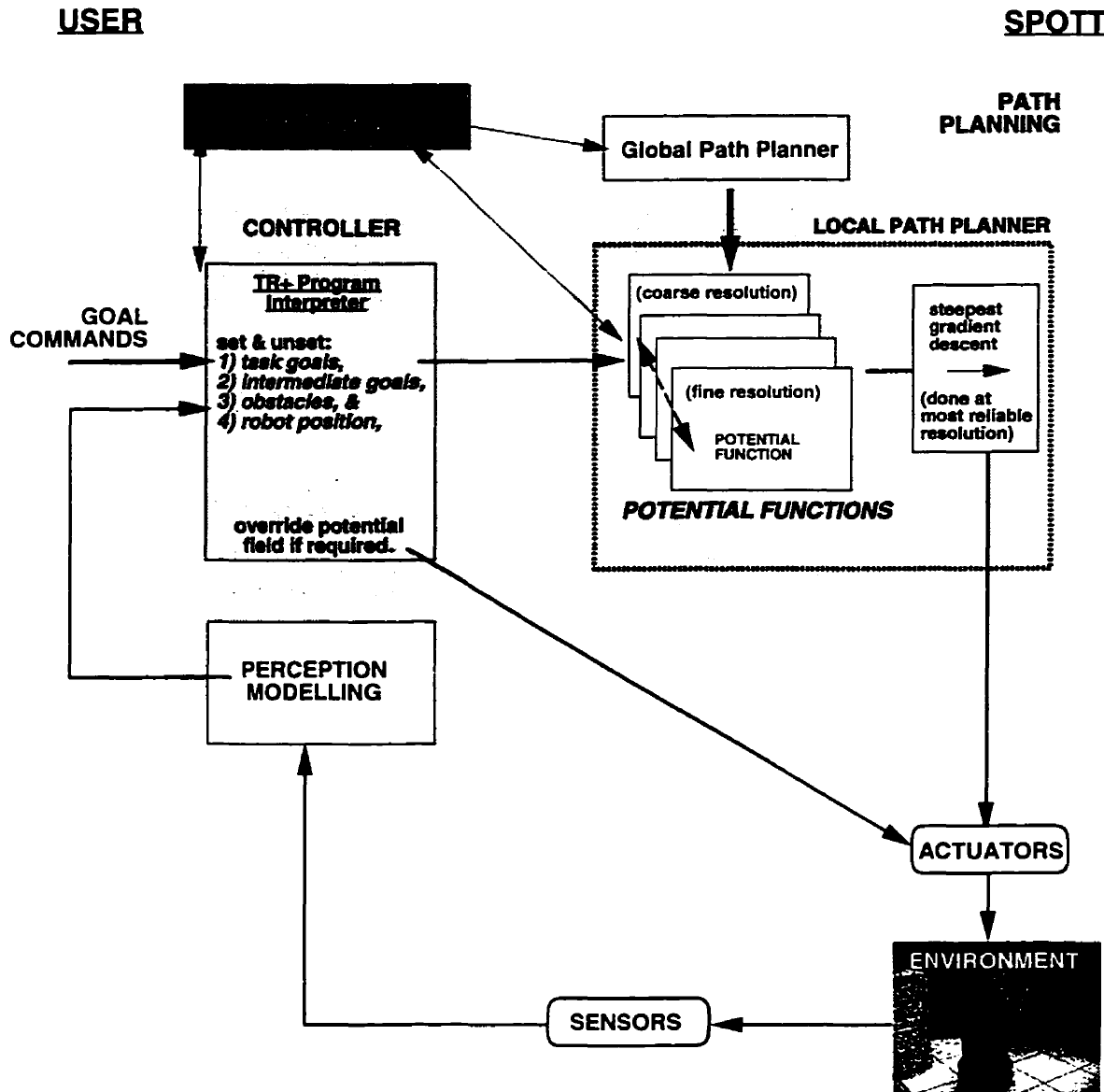


FIGURE 4.1. Path Planning Within SPOTT The goal specification is given as input by the user. If an a priori CAD map exists, it is loaded into the Map Database. The role of the TR+ program is to provide the local path planner (i.e., potential field) with the necessary information for computation, such as obstacle configurations and the current robot position. The TR+ program continually computes this by processing sensor data. As the path is being computed, another module is responsible for concurrently performing steepest gradient descent on the potential function in order to determine the local trajectory for the robot. In order to guarantee proper control, there is a collection of potential functions computed at varying resolutions. The finest resolution potential function that has converged to its solution is used for trajectory generation. The TR+ program can also override this trajectory generation. This usually occurs for safety reasons when there has to be a quick response to sensory stimuli indicating a dangerous situation for the robot.

Continuous navigation is workable if trajectory commands are issued concurrently with the plan computation, or the computation is faster than the rate of change in the environment. SPOTT's local path planner concurrently computes and executes the plan. The



robot controller solicits trajectory commands from the local path planner at the same time the path is being computed. There is no guarantee on the correctness of the trajectory commands before the computation has converged to the solution for the current configuration. In order to address the correctness of the trajectory, a type of “*anytime algorithm*” (Mouaddib & Zilberstein, 1995) is used, which trades off accuracy<sup>1</sup> of the solution versus computation time. A collection of local path planners at varying spatial resolutions are executed. The finer the resolution, the more time the computation requires. It is assumed that the computation speed of the path planner at the coarsest resolution is faster than the rate of trajectory-command requests. The coarsest resolution local path planner is used first, and the finer resolution local path planners will eventually be used once their computations have converged to their respective solutions, provided that there have been no sensed environmental changes. SPOTT’s local path planner uses the potential field approach to implement such a strategy for dynamic path planning.

The potential field approach applies a function over a discrete grid consisting of a configuration of obstacles and goals, and the path is determined by performing gradient descent on this function. SPOTT uses a specific type of potential function called a harmonic function. This function has the desirable property of no local minima (Connolly & Grupen, 1993; Tarassenko & Blake, 1991). If at least one path exists to a known destination, the path planning strategy is guaranteed to find a path to that goal (Doyle & Snell, 1984). A harmonic function is the solution to Laplace’s equation. The important contributions of the described harmonic function potential field approach for local path planning are (1) the guaranteeing of proper control with a collection of potential functions at varying resolutions, (2) limiting the extent of the potential function<sup>2</sup> and sliding its bounds when the robot moves around the environment, and (3) an implementation which separates the computation from the control<sup>3</sup>.

SPOTT is able to guarantee task completion<sup>4</sup> because of the collaboration between the local (potential field) and global (AI-based) path planners. The two planners function in parallel at different time-scales. The local path planner is in a control feedback loop with the robot and environment, and its response time to sensory input is crucial to the real-time operation of the robot. It dynamically reacts to changes in the map database which

<sup>1</sup>The path produced is based on visiting neighboring grid points in a discretized grid. The finer the grid spacing, the more accurate the path.

<sup>2</sup>This is necessary because the harmonic function is a rapidly decaying function and its computation time increases exponentially in direct proportion to the number of grid elements.

<sup>3</sup>Trajectory commands are solicited at the same time the potential function is computed.

<sup>4</sup>SPOTT is able to guarantee task completion when the location of the goal is known and a CAD map is available a priori.

is continually updated by the behavioral controller. On the other hand, the global path planner uses states which change at a slower rate. These states are the current and potential physical locations of the robot with respect to nodes in an abstract graph structure, where a node is a room or a hallway portion. The global planning module advises the local planning module of the local effects of a global goal.

## 1. Path Planning Approaches

Path planning usually requires that a map (e.g., a CAD map of the indoor environment fused with sensor data that is continually being acquired) be transformed into a graph structure, which is then searched for a path. This can be achieved by parcelling the map into equally - or unequally - sized convex<sup>5</sup> polygonal regions<sup>6</sup>. Nodes are used to represent the regions, and edges connect nodes that share part or the whole of a common boundary. Each node and edge is labeled as passable or impassable, and a search is initiated to find a path through passable nodes via passable edges from a start to stop node.

There are three main approaches to path planning (Hwang & Ahuja, 1992) identified in the literature.

- (i) The **roadmap** method captures the connectivity of the robot's free space<sup>7</sup> in a network of one-dimensional curves - called the **roadmap**,  $\tau$  - lying in the free space. Path planning is reduced to connecting the initial and goal configurations to points in  $\tau$  and searching  $\tau$  for a path between these points. Two common roadmaps (also called a skeleton) are the *visibility graph* and the *Vornoi diagram*. The visibility graph is the collection of lines in the free space that connect visible<sup>8</sup> features (e.g., object vertices) to one another. The Vornoi diagram is defined as the set of points that are equidistant from two or more object features.
- (ii) The **cell decomposition** method decomposes the robot's free space into simple regions called cells. A non-directed graph representing the adjacency relation between the cells is then constructed and searched. The adjacency relation is represented in a graph form called the "*connectivity graph*". There are two types of cell decomposition methods.
  - (a) The "*exact cell decomposition*" approach treats the union of all the cells as exactly equal to the free space.
  - (b) In the "*approximate cell decomposition*" approach, the cells are a predefined shape (e.g. rectangles), whose union is strictly included<sup>9</sup> in the free space. It is not necessary for the boundary of a cell to have a physical meaning.

<sup>5</sup>It is not necessary for the regions to be convex, but this is usually the case.

<sup>6</sup>SPOTT's local path planner parcels the map into equally sized square regions (see Section 3.2). Its global path planner parcels the map into larger unequally sized rectangular regions (see Section 8). In the global path planner's representation, the edges represent access ways (e.g., doorways) between nodes.

<sup>7</sup>The set of passable nodes.

<sup>8</sup>Two features are *visible* if a straight line that connects them does not intersect any objects.

<sup>9</sup>The union of all the shape cells is less than or equal to the free space. If part of an object is contained in one of the shape cells, then that shape cell is identified as being occupied.

- (iii) The **potential field method** is a member of the family of “*approximate cell decomposition*” techniques. Its configuration space<sup>10</sup> is discretized into a fine regular grid. For a point robot, the configuration space is identical to the world space<sup>11</sup> (Hwang & Ahuja, 1992). The grid elements, occupied wholly or partially by objects, are saturated positively for obstacles or negatively for goals. The space left after the removal of the objects is the free space. The potential function in the free space<sup>12</sup> has values that are bounded by (but not equal to) the positive and negative saturation values. This grid is searched for a free path. The term *potential field* corresponds to the negated gradient vector field,  $-\nabla U$ , of a potential function  $U$ . The path is usually determined by performing steepest gradient descent on the potential function. The potential function in essence captures all the achievable paths from every possible starting location in the free space. There are no extra computations required if the estimated position of the robot is corrected (i.e., localized).

---

<sup>10</sup>A configuration of an object or robot is a set of independent parameters completely specifying the position of every point of the object or robot. The space of all possible configurations is called the configuration space (C-space).

<sup>11</sup>The world space refers to the physical space in which robots and obstacles exist.

<sup>12</sup>The potential function is only formulated in the free space.

## 2. Dynamic Path Planning Problem

The dynamic path planning problem extends the basic navigation planning problem (Latombe, 1991). The preliminaries and assumptions of the dynamic path planning problem are as follows:

- Let  $R$  be a single rigid object - the robot - moving in a Euclidian space  $E$ , called the environment, represented as  $\mathbb{R}^N$ , with  $N = 2$ . Only 2D motion is considered, however the concepts apply to higher dimensions, where  $N > 2$ .
- Let  $\beta_1, \dots, \beta_n$  be either fixed (or moving rigid) objects distributed in  $E$ . The positions (and dynamics) of an object  $\beta_i$  may or may not be known a priori. The *free space* of the robot is the allowable space for navigation and is obtained by subtracting all the  $\beta_i$  models from the environmental space  $\mathbb{R}^2$ .
- Assume that both the geometry of  $R, \beta_1, \dots, \beta_n$  and the locations of the  $\beta_i$ 's in  $E$  can be approximated to within a certain predefined degree of slack or tolerance. Assume further that the only kinematic constraint that limits the motions of  $R$  is the two-dimensional plane on which it navigates.

Given the above definitions and assumptions, the dynamic path planning (see Figure 4.2) problem is defined as follows:

- Given an initial position and orientation ( $q_{start}$ ) and a goal position and orientation ( $q_{goal}$ ) of  $R$  in  $E$ , generate a path  $\tau$  specifying a continuous sequence of positions and orientations of  $R$ , avoiding contact with the  $\beta_i$ 's, starting at the initial position and orientation, and terminating at the goal position and orientation. As the robot navigates, the  $\beta_i$ 's may be known a priori, appear suddenly, disappear suddenly, remain unchanged, or be dynamic<sup>13</sup> (see Figure 4.2). It can be further assumed that  $R$  is circular and specifying the orientation of  $R$  is not a problem<sup>14</sup>. The orientation of the goal does not necessarily need to be attained. Failure is reported if the goal is unattainable in the current environment, or if the position of  $R$  is lost, or if the obstacles  $\beta_i$ 's in the environment  $E$  cannot be monitored with sufficient detail.

<sup>13</sup>However, at this moment SPOTT is only able to treat the case where objects suddenly appear. If the location of appearance is not where a current object is situated, the model associated with the newly discovered object is added to the map database. Extending SPOTT's local path planner to include dynamic object models is an issue for future research.

<sup>14</sup>The robot can be modelled as a point, as long as it is symmetrical and circular. In order to compensate for this, all features in the environment are padded by a size equal to half the robot's diameter (i.e., 26 cm) in all directions (i.e., a line becomes a rectangle). In addition, to compensate for the uncertainties associated with the position of the sensed features and the robot, a potential error factor which is equal to one quarter the diameter of the robot (i.e., 13 cm) is added to all obstacles in all directions.

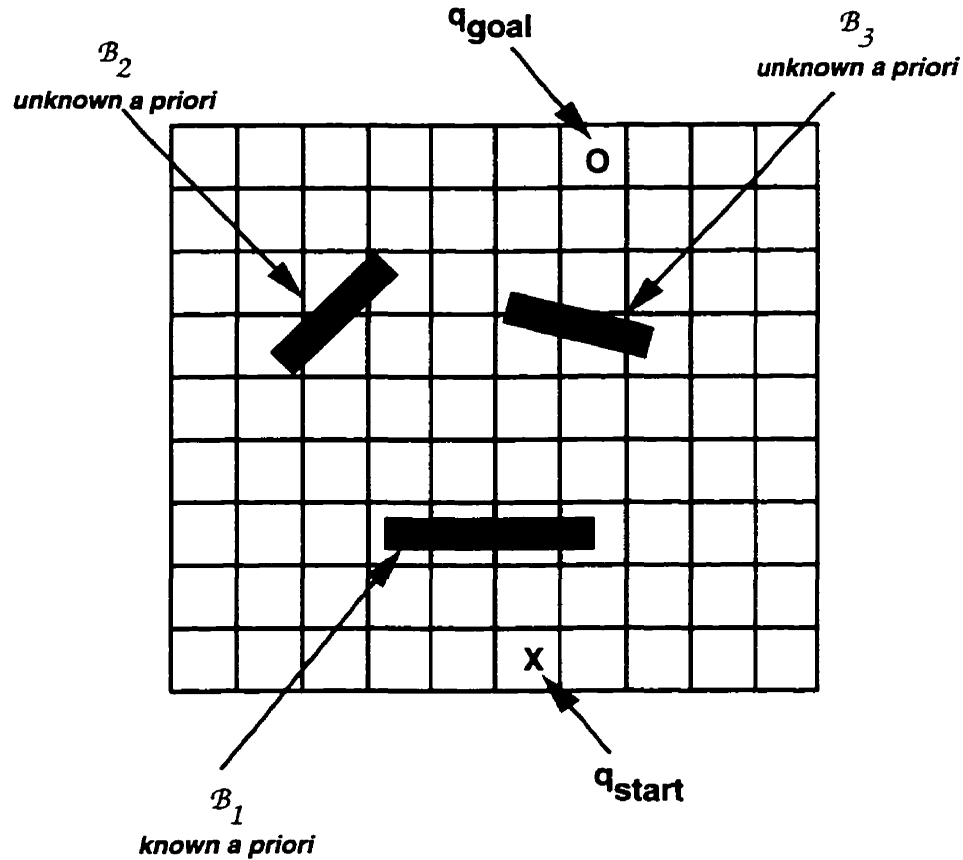


FIGURE 4.2. **Dynamic Path Planning Problem.** Given an initial position and orientation ( $q_{start}$ ) and a goal position and orientation ( $q_{goal}$ ) of  $R$  in  $E$ , generate a path  $\tau$  specifying a continuous sequence of positions and orientations of  $R$  avoiding contact with the  $\beta_i$ 's, starting at the initial position and orientation, and terminating at the position and orientation of the goal. As the robot navigates, the  $\beta_i$ 's may be known a priori, appear suddenly, disappear suddenly, remain unchanged, or be dynamic.

There may be many different paths from the initial starting position to the goal. The task is to attain a plausible path, which may be the optimal path, but not necessarily<sup>15</sup>.

<sup>15</sup>SPOTT's local path planner uses a potential field approach. A harmonic function is used as the potential function and it is equivalent to a probability function representing random walks on a Markov chain (Doyle & Snell, 1984). It can also be interpreted as representing a hitting probability (Connolly, 1994). Steepest gradient descent on this function produces an optimal path in the sense that the robot is positioned at locations which maximally reduce the hitting probability (see Section 5).

### 3. Path Planning Using Potential Fields

A potential function is formulated in the *free space* such that obstacles appear to exert negative potentials and goals, positive potentials. It can either be a heuristic combination of functions or a single function superimposed onto the configuration space. In the heuristic approach, the potential function is the sum of a collection of potential functions emanating from the obstacle and goal models (Khatib, 1986). The potential function emanating from an obstacle is usually modelled as a symmetrically decaying function (i.e., Gaussian,  $\frac{1}{r}$ ,  $\frac{1}{r^2}$ ), and the potential function radiating from a goal is a symmetrically increasing function<sup>16</sup>. An example of the second approach is to compute a harmonic function over the free space. This function is a solution to Laplace's equation and is used to model various physics-based problems such as laminar fluid flows around solid obstacles and current flow in a conducting medium interspersed with nonconducting holes.

#### 3.1. Biological Inspiration

The potential function has been used to model biological behavior associated with navigational path planning. Arbib (1987) developed a neural model to try to explain the behavior of a frog when presented with a worm as a goal and a fence as an obstacle. The models were used to explain visual orientation and the choice of trajectory movement, and both took on a form similar to the usual potential field. Physiological studies in primates have also found various neural representations - similar to potential fields - in the subcortical areas of the brain which encode saccades<sup>17</sup> and covert attentional shifts (Robinson & Petersen, 1992). Connolly and Burns (1992) suggest that another subcortical area in the brain - the basal ganglia - may be an important locus for the mechanisms of harmonic function computation. They hypothesize that this function may be used in the planning of goal-oriented and obstacle-avoiding behavior in biological systems.

#### 3.2. Formulating a Potential Function

Every object  $\beta_i$ ,  $i = 1$  to  $n$ , in the workspace  $E$  maps into a region of  $C$  (the configuration-space). The objects are modelled in 2D space as either points, lines, ellipses, or rectangles<sup>18</sup>. The union of all the object models is called the *object space*:

$$(4.4.1) \quad C_{object} = \bigcup_{i=1}^n \beta_i$$

<sup>16</sup>Usually the inverse of the function used for obstacles.

<sup>17</sup>Sudden eye movements.

<sup>18</sup>Other two-dimensional models may also be used, such as an n-sided polygon.

The region specified by removing the union of all object model regions is called the *free space*:

$$(4.4.2) \quad C_{free} = C - C_{object} = C - \bigcup_{i=1}^n \beta_i$$

Let  $p_{max}$  be the maximum value of the potential function, and  $p_{min}$  be the minimum value of the potential function. These are referred to as the saturation values. Let  $e$  be the discretization of the potential field space (in 2D,  $e = (x, y)$ ). All object model regions, whether obstacles or goals, are represented as saturated values in the potential function:

$$(4.4.3) \quad C(x, y) = \begin{cases} p_{max}, & (x, y) \in \beta_i, \beta_i = \text{obstacle} \\ p_{min}, & (x, y) \in \beta_i, \beta_i = \text{goal} \end{cases}$$

The potential function is defined for every point (i.e., grid element) in the free space. The potential field is derived by taking the gradient at every point in the potential function. The gradient represents the velocity vector which is used to locally control the robot. The net result of a goal model is to generate an *attractive potential* which pulls the robot towards the goal. In contrast, an obstacle model produces a *repulsive potential* which pushes the robot away from the obstacle.

### 3.3. Summation of Potentials Approach

In the summation of potentials approach, the potential function is created by summing together attractive potential functions for each goal model and a repulsive potential function for each obstacle model.  $U_{att}$  is called the attractive potential associated with a collection of goal configurations  $\beta_{goal}$ :

$$(4.4.4) \quad U_{att}(x, y) = \sum_{i=1}^g (U_{att_i}(x, y))$$

$U_{rep}$  is called the repulsive potential associated with a collection of obstacle configurations ( $\beta_{obstacle}$ ):

$$(4.4.5) \quad U_{rep}(x, y) = \sum_{j=1}^{n, j \neq goal} (U_{rep_j}(x, y))$$

Let  $U$  be the artificial potential function in the free space  $C_{free}$ , constructed as the sum of obstacle and goal potential functions:

$$(4.4.6) \quad U(x, y) = U_{att}(x, y) + U_{rep}(x, y)$$

The elementary potential function for an obstacle is in the form of a decaying function emanating from the obstacle's geometric model. Examples of such a function are a Gaussian,



$\frac{1}{r}$ , or  $\frac{1}{r^2}$  (where  $r$  is the distance from the object model). The attraction potential function is a mirror image of the repulsive potential function:

$$(4.4.7) \quad U_{att}(x, y) = -U_{rep}(x, y)$$

The potential function created using the summation of potentials approach can have a local maximum or minimum in places that are not desirable. A local maximum is only appropriate in locations that correspond to obstacle models (i.e., outside of the free space), and a local minimum should only occur at the location of the goal model. When two object models are closely positioned, the summation of their emanating potentials creates a local maximum (see Figure 4.3). Similarly, a local minimum can occur when summing two closely positioned goal potentials.

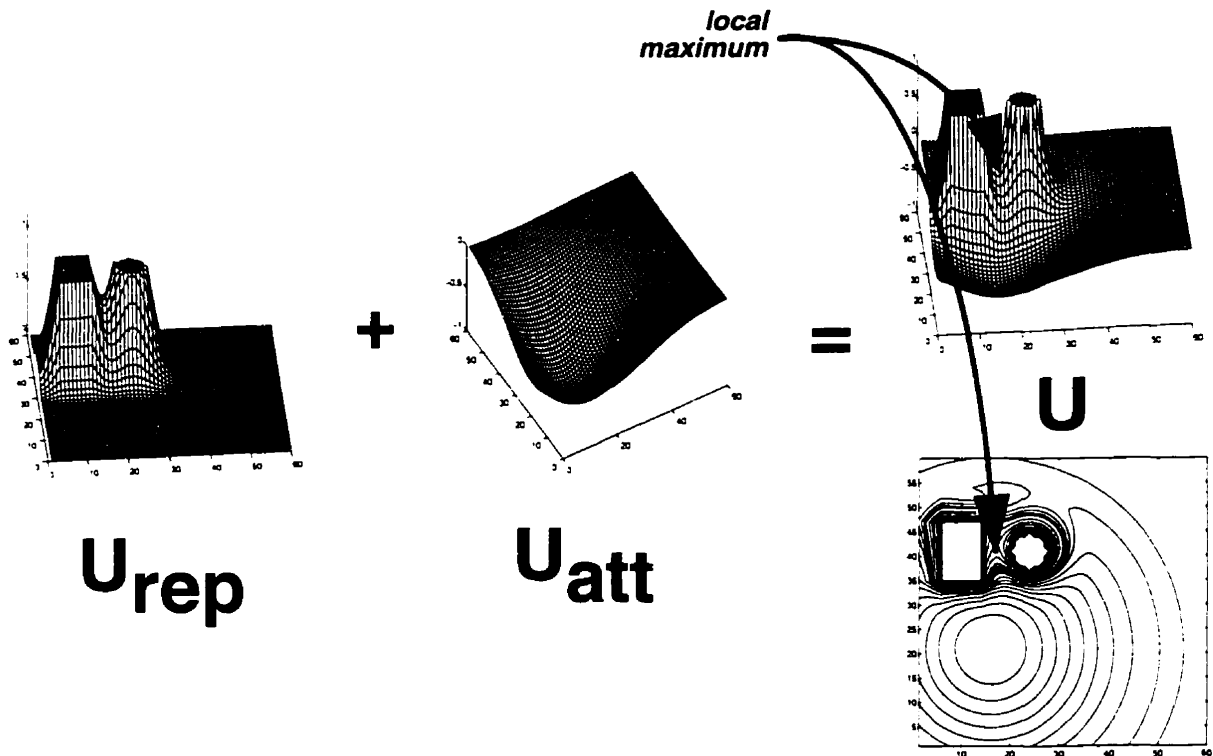


FIGURE 4.3. Summation of Potentials: Close Objects. The potential function  $U$  is created by summing the obstacle potentials  $U_{rep}$  with the goal potentials  $U_{att}$ . When two obstacles are closely positioned, the potential resulting from the summation creates a local maximum, as shown above.

A method for minimizing this phenomenon is to use the maximum value as opposed to the summation when combining obstacle potentials, and to use the minimum when

combining goal potentials:

$$(4.4.8) \quad \begin{aligned} U_{rep}(x, y) &= \max_{j=1}^{n, j \neq goal} (U_{rep_j}(x, y)) \\ U_{att}(x, y) &= \min_{i=1}^g (U_{att_i}(x, y)) \end{aligned}$$

See Figure 4.4 for an example of a potential function created using equation 4.4.8. Equation 4.4.8 does not guarantee that no spurious local minima will occur, but reduces their probability of occurrence. The calculations required are also no more complex than those in Equations 4.4.4 and 4.4.5.

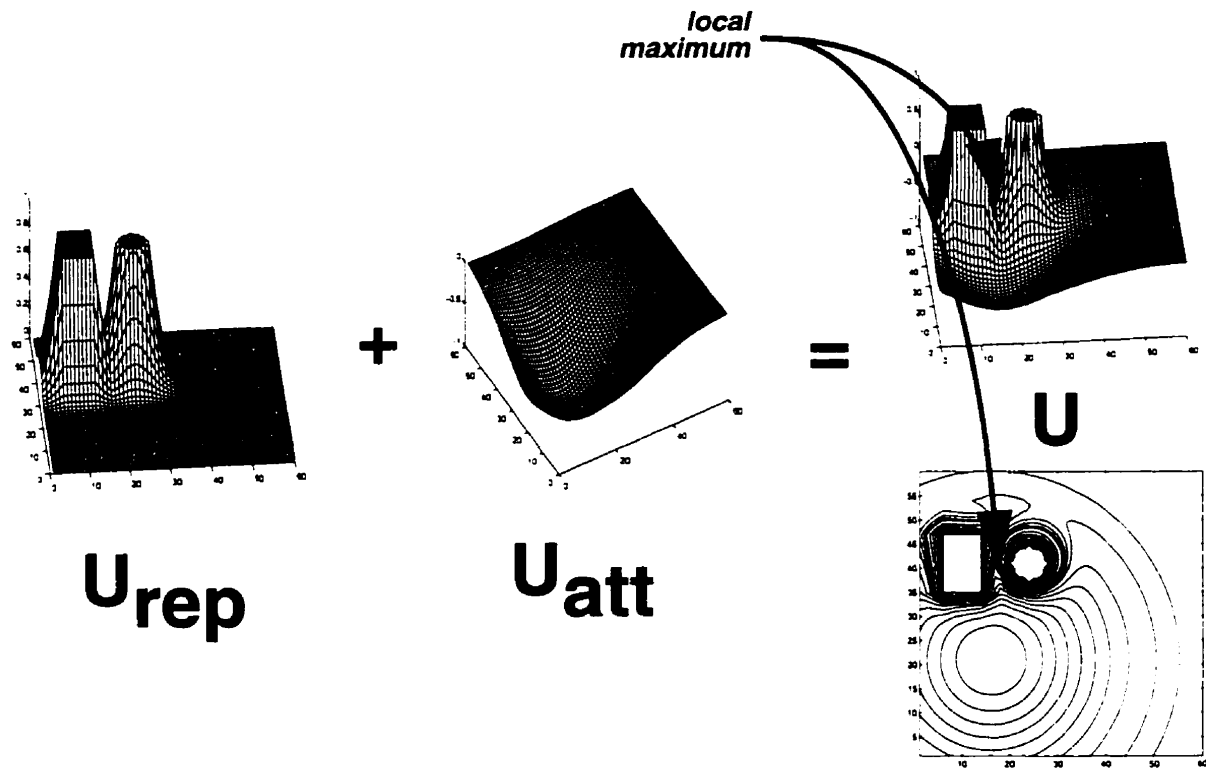


FIGURE 4.4. Maximum Summation of Potentials: Close Objects. The potential function is created by summing the obstacle potentials with the goal potential. The obstacle potential is created by taking the maximum value of all contributing obstacle potentials. Even with this combination strategy, the potential resulting from the summation creates a local maximum, when two objects are closely positioned. The maximum is less than the one shown in Figure 4.3, but it still exists.

The advantages of using a heuristic potential approach (i.e., such as the summation of potentials approach) is that the computations are simple enough so that it can be implemented as a real-time planning procedure. Path planning is executed by performing gradient descent on the potential function. In general, the potential field method removes the need to reason about the topological relations of objects, as this is inherent in the potential field control mechanism. The disadvantage of using the heuristic potential function

as the sole control mechanism is that local minima - other than the goal minimum - may be present.

There are several methods of path planning which use the heuristic potential function created by the summation of potentials approach (Latombe, 1991). The *depth-first planning* technique is essentially the steepest gradient descent method. It escapes local minima using heuristic methods, which may not always succeed. A simple heuristic action to take is to traverse around the object's perimeter (Choi & Latombe, 1991). The *best-first planning* technique uses a steepest descent method; however when a local minimum is reached, the planner backtracks out of the local minimum and follows the next best potential gradient. The *variational* approach employs a functional  $J$  of a path  $\tau$  and this is optimized over all possible paths. The *best-first planning* and *variational* techniques are guaranteed to find a solution at the expense of a potentially costly search which includes backtracking. This is in direct contrast to the inexpensive execution cost associated with the steepest gradient descent method. It is fast because it asks for very little interpretation of the potential function.

Other ways of dealing with local minima are to alter the potential function so as to minimize or eliminate local minima. Some researchers (Arkin, 1993; Gat, 1992) minimize local minima by specifying goals as a set of piecewise linear trajectories. In a similar fashion, Krogh and Thorpe (1986) find a sequence of critical points along a globally observed path, which are first computed and set as goals in a potential function. The critical points are selected so that the chances of getting stuck in a local minimum are minimized.

Another method of avoiding local minima is to control the potential function, by using spherical models for the objects (Rimon & Koditschek, 1990). If all objects are modelled as spherical regions, the problem is to construct a diffeomorphism that deforms a given set of obstacles into a set of spherical regions. However, this is not always possible.

The existence of local minima is a concern when using a heuristic potential function as the input to a path planner. Ideally, it is desirable to have a potential function that has no local minima. This type of function can then be safely used for trajectory control by using the simple steepest gradient descent method.

#### 4. Harmonic Functions as Potential Functions

A potential function which has the desirable property of no local minima is the *Harmonic function* (Connolly & Grupen, 1993; Tarassenko & Blake, 1991). If at least one path exists to a known destination location, steepest gradient descent on this function is guaranteed to find it (Doyle & Snell, 1984).

A harmonic function on a domain  $\Omega \subset R^n$  is a function which satisfies Laplace's equation:

$$(4.4.9) \quad \nabla^2 \phi = \sum_{i=1}^n \frac{\delta^2 \phi}{\delta^2 x_i^2} = 0$$

The value of  $\phi$  is given on a closed domain  $\Omega$  in the configuration space  $C$ . A harmonic function satisfies the *Maximum Principle* and *Uniqueness Principle* (Doyle & Snell, 1984). The *Maximum Principle* guarantees that there are no local minima in the harmonic function.

**DEFINITION 1 (Maximum Principle).** *A harmonic function  $f(x, y)$  defined on  $\Omega$  takes on its maximum value  $M$  and its minimum value  $m$  on the boundary.*

The *Uniqueness Principle* guarantees that there is a unique solution to Laplace's equation for a given configuration.

**DEFINITION 2 (Uniqueness Principle).** *If  $f(x, y)$  and  $g(x, y)$  are harmonic functions on  $\Omega$  such that  $f(x, y) = g(x, y)$  for all boundary points, then  $f(x, y) = g(x, y)$  for all  $x, y$ .*

The harmonic function has two physical analogies. In the first instance, obstacles are modelled as nonconducting solids in a conducting medium. The boundaries of the obstacles and  $\Omega$  are current sources and the goal is an equal and opposite current sink (Tarassenko & Blake, 1991). Integral lines of the current field form feasible paths for navigation. Alternatively, obstacles are modelled as solids immersed in a laminar fluid flow. The feasible paths are the streamlines of the fluid flow.

##### 4.1. Computing Harmonic Functions

Laplace's equation is solved in a two-dimensional domain when used as a potential function for navigational path planning on a 2D planar surface. The domain is defined by a local extent and a collection of obstacle and goal models. Navigation is confined to the free space which is delineated by the obstacle and goal model boundaries and the closed domain boundary of  $\Omega$ . The harmonic function can be iteratively computed by taking advantage of its inherent averaging property: the value of any point in the free space is equivalent to the average of its neighbouring points. Numerical solutions for Laplace's equation are obtained

from finite difference equations (Hornbeck, 1975). The following expression for Laplace's equation is obtained by combining a Taylor series expansion for a forward difference and backward difference equation:

$$\begin{aligned}
 \nabla^2 u|_{i,j} &= u_{xx}|_{i,j} + u_{yy}|_{i,j} \\
 (4.4.10) \quad &= (u_{i+1,j} + u_{i-1,j} - 2u_{i,j}) + (u_{i,j+1} + u_{i,j-1} - 2u_{i,j}) + O(h^2) \\
 &= u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} + O(h^2)
 \end{aligned}$$

The sampling ratio is  $h$ , where  $h = \Delta x = \Delta y$ . If  $n_x$  and  $n_y$  are the dimensions of  $\Omega$ , then  $\Delta x = \frac{1}{n_x}$  and  $\Delta y = \frac{1}{n_y}$ . When Laplace's equation -  $u_{xx} + u_{yy} = 0$  - is approximated using Equation 4.4.10, the error term -  $O(h^2)$  - is omitted, resulting in the implicit relation<sup>19</sup>:

$$(4.4.11) \quad U_{i,j} = \frac{1}{4}(U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1})$$

Equation 4.4.11 can be used to perform an iterative computation subject to a given set of boundary conditions. This computation converges to the harmonic function which is the solution to Laplace's equation.

A more accurate expression for Laplace's equation can be obtained in a similar fashion, utilizing the neighbouring diagonal elements (van de Vooren & Vliethart, 1967):

$$\begin{aligned}
 \nabla^2 u|_{i,j} &= \frac{1}{5}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}) \\
 (4.4.12) \quad &+ \frac{1}{20}(u_{i+1,j+1} + u_{i-1,j+1} + u_{i-1,j-1} + u_{i+1,j-1}) \\
 &+ u_{i,j} + O(h^8)
 \end{aligned}$$

In Equation 4.4.12, the error term is a lot smaller than in Equation 4.4.10. Omitting the error term, the implicit relation for the nine-point kernel is written as follows:

$$\begin{aligned}
 (4.4.13) \quad U_{i,j} &= \frac{1}{5}(U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1}) \\
 &+ \frac{1}{20}(U_{i+1,j+1} + U_{i-1,j+1} + U_{i-1,j-1} + U_{i+1,j-1})
 \end{aligned}$$

The nine-point kernel gives smoother<sup>20</sup> intermediate results during the iterative computation as compared to the five-point kernel.

The computation of the harmonic function can be formulated with two different types of boundary conditions. The *Dirichlet* boundary condition<sup>21</sup> is where  $u$  is given at each point of the boundary. The *Neumann* boundary condition is where  $\frac{\partial u}{\partial n}$ , the normal component of the gradient of  $u$ , is given at each point of the boundary. In order to have flow, there has to be a source and a sink. Thus, the boundary of the mesh is modelled as a source, and the boundary of the goal model is modelled as a sink. The boundaries of the

<sup>19</sup>The exact result at node  $(i, j)$  is denoted by  $u_{i,j}$ , and the approximate value by  $U_{i,j}$ .

<sup>20</sup>This is because the error term is smaller in formulating the nine-point kernel compared to the five-point kernel.

<sup>21</sup>This inherently makes all applicable boundary points into sources (i.e., in terms of sources and sinks for modelling liquid flow).

obstacles are modelled according to the type of boundary condition chosen: Dirichlet or Neumann. If the obstacle boundaries are modelled using the Dirichlet boundary condition, then they are also considered sources. Otherwise (i.e., for a Neumann boundary condition), only the gradient at the obstacle boundary is defined and it is not a source. The harmonic function has different properties which depend on the type of boundary condition used in its formulation. The solution to a Dirichlet boundary condition problem has exponential decay emanating from the goal. This causes some values to be unexpressible in a memory address of a thirty-two-bit computer (Tarassenko & Blake, 1991). This is especially evident with long corridor features. In this case, the maximum computable length-to-width ratio for a thirty-two-bit computer is approximately 7.1 to 1 (see Appendix A for the derivation). The Neumann boundary condition causes steepest gradient-descent paths to graze the obstacle boundaries (see Figure 4.5). This is because for Neumann boundary conditions, an obstacle's neighbouring points do not depend on the obstacle boundary point's value in their calculation. For example, if  $U_{i,j}$  is the bordering point, and  $U_{i,j+1}$  is the boundary point, the Neumann condition implies that the gradient in the direction of the bordering point -  $U_{i,j} - U_{i,j+1}$  - is zero. Inserting this condition in Equation 4.4.11 results in the following equivalence for the bordering point:  $U_{i,j} = \frac{1}{3}(U_{i-1,j} + U_{i+1,j} + U_{i,j-1})$ . Therefore,  $U_{i,j+1}$  has no influence on the value of  $U_{i,j}$ . This has the effect<sup>22</sup> of pulling the steepest descent gradient path towards nearby obstacles. In contrast, the Dirichlet boundary condition results in paths which are smooth and do not brush up against obstacle boundaries (see Figure 4.6). Non-grazing robot trajectories are more desirable because this permits a greater tolerance for errors in the modelling of object boundaries. It has been suggested to use a combination<sup>23</sup> of the Dirichlet solution and the Neumann solution (Connolly & Grupen, 1993). However, this will not override the problem of properly representing the Dirichlet solution in a thirty-two-bit computer, as the Dirichlet component's effect on the solution will be negligible in long corridor environments. The Neumann solution can also have a rapidly vanishing field problem, but the exponential decay is slower than the Dirichlet solution. Limiting the extent of the potential function boundary (i.e., making it a local path planner) permits the use of Dirichlet boundary conditions.

There are three ways of speeding up an iterative computation of a harmonic function: (1) speeding up the time for a single iteration; (2) reducing the number of iterations; and

<sup>22</sup>The reason why an obstacle attracts a steepest gradient descent path when using the Neumann boundary condition can be shown by observing the equivalence of the equation  $U_{i,j} = \frac{1}{3}(U_{i-1,j} + U_{i+1,j} + U_{i,j-1})$  to Equation 4.4.11 multiplied by  $\frac{4}{3}$  with  $U_{i,j+1}$  set to 0 (i.e., a goal attractor).

<sup>23</sup>The linear combination of harmonic functions is also a harmonic function (Axler *et al.*, 1991). Harmonic properties are also preserved under dilation, translation, scaling and with applying a bias.

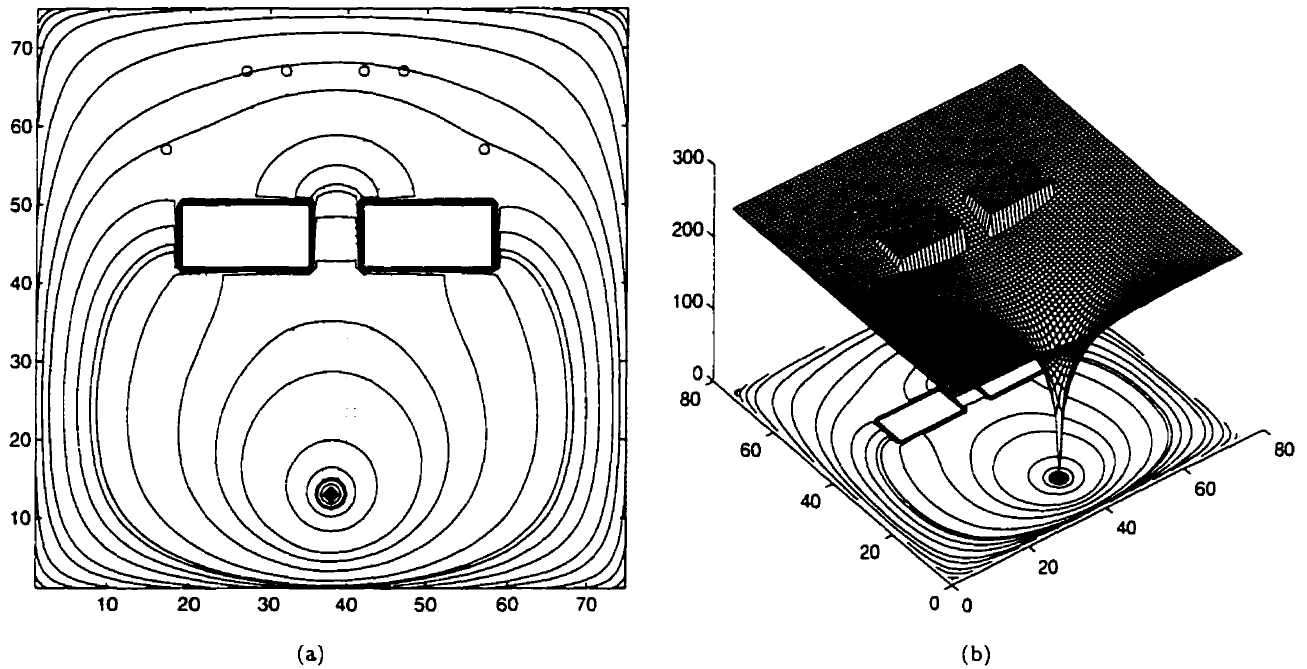


FIGURE 4.5. Potential Field Using Neumann boundary conditions. (a) shows the equal-potential contour plot of the potential function and various generated paths using steepest gradient descent from different starting points. The paths closely follow the obstacle boundaries. (b) shows the mesh diagram of the potential function. The obstacles are distinguishable from the rest of the potential because for Neumann boundary conditions, an obstacle's neighbouring points do not depend on the obstacle boundary point's value in their calculation.

(3) providing a good initial guess to the solution when initiating the computation for a particular configuration. The time for a single iteration can be reduced by parallelizing the computation (see Chapter 6 for a discussion of the implementation issues). There is a collection of techniques called “*Methods-of-Relaxation*” (Hornbeck, 1975; Ames, 1992) which reduce the number of iterations. A good initial guess to the computation can be obtained by using the summation of potentials technique which was discussed earlier in Section 3.3.

#### 4.1.1. Number of Iterations Reduced by Methods-of-Relaxation

The simplest method-of-relaxation is called the *Jacobi* method<sup>24</sup>. It has a slow convergence rate and is rarely used. The basic single step of the iteration consists in replacing the current value  $U^{(k-1)}$  by the improved value<sup>25</sup>  $U^{(k)}$ , which is obtained as follows for the

<sup>24</sup> Also called “*iteration by total steps*”, and the “*method of simultaneous displacements*”.

<sup>25</sup>  $U_{i,j}^{(k)}$  is written such that the superscript -  $k$  - is the iteration index and the subscript -  $i, j$  - is the spatial position  $(i, j)$ .

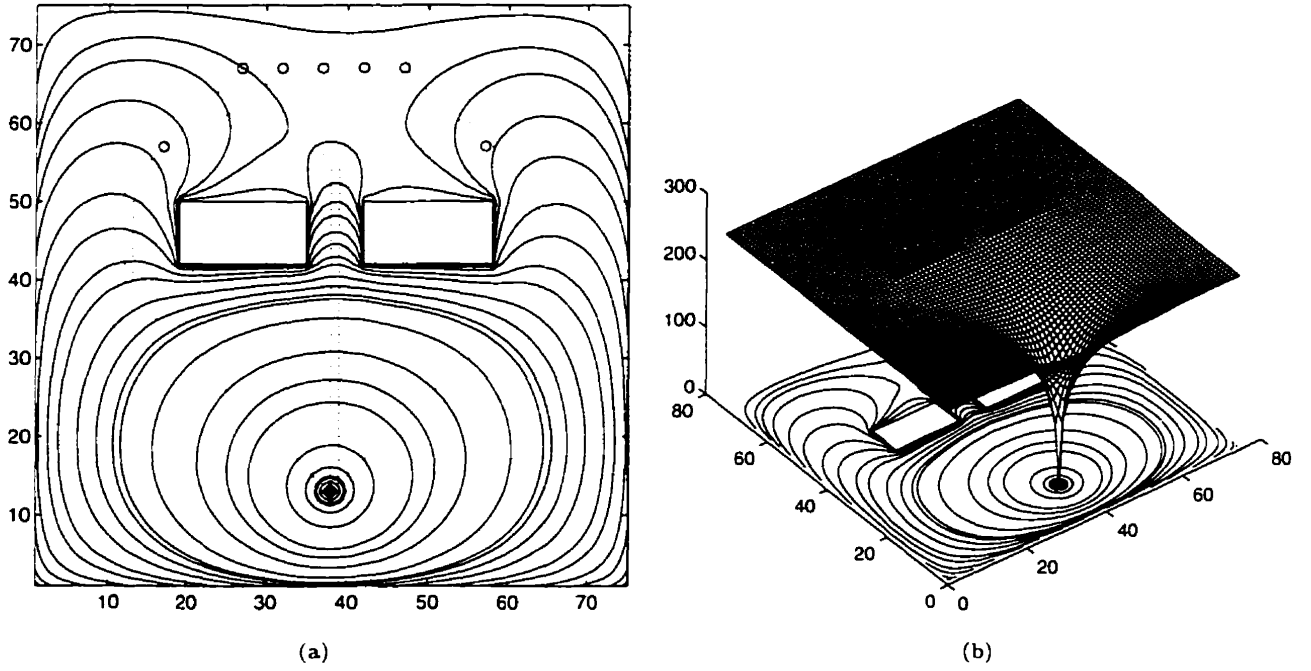


FIGURE 4.6. Potential Field Using Dirichlet boundary conditions. (a) shows the equal-potential contour plot of the potential function and various generated paths using steepest gradient descent from different starting points. The paths do not graze the obstacle boundaries. (b) shows the mesh diagram of the potential function. The obstacles are barely distinguishable from the rest of the field in the mesh diagram because the obstacle's neighbouring points are near the saturated value that represents the obstacles. This can be explained by viewing the potential values, when using Dirichlet boundary conditions, as representing hitting probabilities (i.e., the probability of hitting an obstacle before arriving at a goal when initiating a random walk from the location in question) (Doyle & Snell, 1984; Connolly, 1994). The hitting probability of a point near an obstacle boundary will be very close in value to the hitting probability of an obstacle point.

five-point kernel:

$$(4.4.14) \quad U_{i,j}^{(k)} = \frac{1}{4}(U_{i+1,j}^{(k-1)} + U_{i-1,j}^{(k-1)} + U_{i,j+1}^{(k-1)} + U_{i,j-1}^{(k-1)})$$

Similarly, the basic single step of the iteration can be written as follows for the nine-point kernel:

$$(4.4.15) \quad U_{i,j}^{(k)} = \frac{1}{5}(U_{i+1,j}^{(k-1)} + U_{i-1,j}^{(k-1)} + U_{i,j+1}^{(k-1)} + U_{i,j-1}^{(k-1)}) + \frac{1}{20}(U_{i+1,j+1}^{(k-1)} + U_{i-1,j+1}^{(k-1)} + U_{i-1,j-1}^{(k-1)} + U_{i+1,j-1}^{(k-1)})$$

This type of computation can be used on a SIMD<sup>26</sup> machine, where an iteration over the grid is achieved in one parallel step using only one solution variable per processing element. Similarly, it also lends itself to an implementation using a resistor lattice (McCann & Wilts, 1949; Tarassenko & Blake, 1991) (see Figure 4.7).

<sup>26</sup>Single Instruction, Multiple Data.



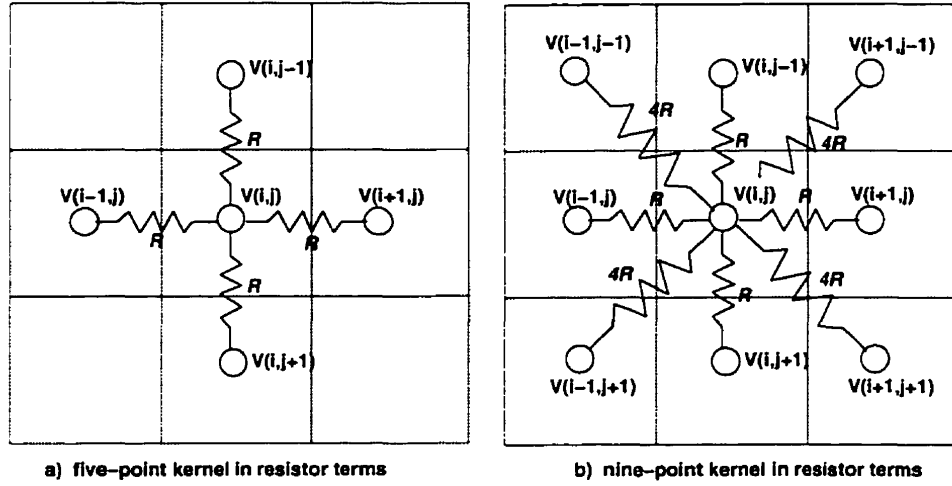


FIGURE 4.7. Iteration Kernel and Resistive Grid. The illustration shows how a resistive grid can be used to implement the iteration kernel which is used in the Jacobi method for both the five-point and nine-point kernels.

The resistor lattice equivalence is obtained by placing a resistor between neighbouring nodes and having the value of a node represent a voltage. Ohm's law states that if points  $a$  and  $b$  are connected by a resistance of magnitude  $R$ , then the current  $I_{a,b}$  that flows from  $a$  to  $b$  is expressed as follows:

$$(4.4.16) \quad I_{a,b} = \frac{V(a) - V(b)}{R}$$

By Kirchoff's Laws, the current flowing into a point defined by the coordinates  $(i, j)$  must be equal to the current flowing out:

$$(4.4.17) \quad \frac{V(i+1,j) - V(i,j)}{R} + \frac{V(i-1,j) - V(i,j)}{R} + \frac{V(i,j+1) - V(i,j)}{R} + \frac{V(i,j-1) - V(i,j)}{R} = 0$$

Multiplying through by  $R$  and solving for  $V(i, j)$  results in the following relationship:

$$(4.4.18) \quad V(i, j) = \frac{1}{4}(V(i+1, j) + V(i-1, j) + V(i, j+1) + V(i, j-1))$$

The equivalence of Equation 4.4.18 and Equation 4.4.11 shows that the Jacobi method can be solved by a resistor lattice. A similar correspondence can also be obtained for the nine-point kernel.

A programmable resistive grid is described and simulated by Tarassenko and Blake (1991). It consists of an array of transistors whose drain-source resistances are governed by gate voltages. The transistor network can be used to set both Dirichlet and Neumann obstacle types. After settling, the voltages at the *free* junctions represent a discrete sampling of a harmonic function. A group at the University of Massachusetts (Stan *et al.*, 1994)

implemented two CMOS VLSI implementations of this grid. One chip implemented the harmonic function computation using Dirichlet boundary conditions, and the other used Neumann boundary conditions. The chip sizes were 16 by 16 nodes and 18 by 18 nodes. It was estimated that for grid sizes of 100 by 100 using a resistive VLSI approach, the harmonic function could be computed in a few microseconds (Connolly & Grupen, 1993), however this has never been physically realized. The two main problems with the VLSI implementation were: (1) the network size was limited because the voltage between any two adjacent nodes became so small that it was buried beneath the noise; and (2) the memory containing the obstacle and goal configuration required to be entirely reloaded when the configuration changed. The University of Massachusetts group, due to the above mentioned complications and technological limitations, have abandoned the VLSI route. They<sup>27</sup> concluded that at this time, a digital approach shows more promise for larger grids, higher dimensions, programmability, testing and reliability.

The *Gauss-Seidel* method<sup>28</sup> of relaxation is based upon the immediate use of the improved values. In order to compute the iterative solution on a computer, the array of data is scanned in a sequential fashion: usually from the top left hand corner, row by row, and in each row, column by column. This imposes an ordering on the computation of mesh points. The newest value of the already scanned grid elements can be used in the computation of subsequent grid elements. The Gauss-Seidel method improves the rate of convergence when compared to the Jacobi method<sup>29</sup>. The Gauss-Seidel iteration step for the five-point kernel is expressed by:

$$(4.4.19) \quad U_{i,j}^{(k)} = \frac{1}{4}(U_{i+1,j}^{(k-1)} + U_{i-1,j}^{(k)} + U_{i,j+1}^{(k-1)} + U_{i,j-1}^{(k)})$$

The Gauss-Seidel iteration step can also be written as follows for the nine-point kernel:

$$(4.4.20) \quad U_{i,j}^{(k)} = \frac{1}{5}(U_{i+1,j}^{(k-1)} + U_{i-1,j}^{(k)} + U_{i,j+1}^{(k-1)} + U_{i,j-1}^{(k)}) + \frac{1}{20}(U_{i+1,j+1}^{(k-1)} + U_{i-1,j+1}^{(k-1)} + U_{i-1,j-1}^{(k)} + U_{i+1,j-1}^{(k-1)})$$

The number of iterations can often be substantially reduced by an extrapolation process from previous iterations of the Gauss-Seidel method (Ames, 1992). This method is called "*Successive Over-Relaxation*" (SOR). The SOR method proceeds as the Gauss-Seidel iteration; however, an extra step is performed before accepting the new value. Let  $\bar{U}_{i,j}^k$  be a component of the k'th Gauss-Seidel iteration. The SOR technique is defined by the

<sup>27</sup>This is based on personal communication with Wayne P. Burses at the University of Massachusetts at Amherst (May 11, 1994).

<sup>28</sup>Also called the "*successive displacements*" or "*iteration by single steps*".

<sup>29</sup> $O(n^2)$  as opposed to  $O(2n^2)$

following relation:

$$(4.4.21) \quad U_{i,j}^{(k)} = (1 - \lambda)U_{i,j}^{(k-1)} + \lambda \bar{U}_{i,j}^k$$

The accepted value at step  $k$  is extrapolated from the Gauss-Seidel value and the previously computed value. If  $\lambda = 1$ , the method reduces to that of Gauss-Seidel. The quantity  $\lambda$  is called the relaxation parameter. The relaxation parameter determines the convergence rate and is a value between 1 and 2. The optimum over-relaxation factor for the problem,  $\lambda_{opt}$ , is estimated by the following computation (Hornbeck, 1975):

$$(4.4.22) \quad \lambda_{opt} = \frac{2}{1 + \sqrt{1 - \omega^2}}$$

where  $\omega$  is determined as follows, when  $k \rightarrow \infty$ :

$$(4.4.23) \quad \frac{\|Y^{(k)}\|}{\|Y^{(k-1)}\|} \rightarrow \omega^2$$

$$(4.4.24) \quad \|Y^{(k)}\| = \sum_{i=1}^n \sum_{j=1}^N |Y_{i,j}^{(k)}|$$

$$(4.4.25) \quad Y_{i,j}^{(k)} = U_{i,j}^{(k)} - U_{i,j}^{(k-1)}$$

In order for the SOR technique to function properly, it is important that the estimate of  $\omega^2$  settle down to a constant value less than 1, before using equation 4.4.22 to estimate  $\lambda_{opt}$ . For a better rate of convergence, it is better to overestimate  $\lambda_{opt}$  than to underestimate it (Hornbeck, 1975).

The relaxation techniques can be evaluated based on the work required per iteration and the number of iterations necessary for convergence. All the relaxation techniques are essentially equivalent with respect to the amount of work per iteration. The time per iteration can be reduced by allotting different regions of the array to different processors, as discussed in Chapter 6. The rate of convergence for the Jacobi method is  $O(2n^2)$ , for a square grid of  $n$  elements, while the rate of convergence for the Gauss-Seidel method is  $O(n^2)$ . The rate of convergence for the optimum SOR method is  $O(n)$  (Ames, 1992). The rates of convergence vary slightly when different types of boundary conditions are used.

Convergence is also improved by following the first sequence in a row direction with a second in the column direction. This method is called an “*Alternating Direction Implicit*” (ADI). It has been shown that this method can improve convergence by a factor of 2 (Ames, 1992).

In order to reduce the number of iterations as much as possible, Gauss-Seidel iteration with Successive Over-Relaxation, combined with the “*Alternating Direction*” (ADI) method is used by SPOTT’s local path planner.

#### 4.1.2. A Good Initial Guess

Another way of speeding up the computation of the harmonic function is to provide the computation with a good initial guess for a given obstacle and goal configuration. A good initial estimate can be provided by the “*summation of potentials approach*” (see Section 3.3).

The initial guess is different depending on when it is applied:

- When the harmonic function iterative computation is initiated with a given CAD map, the initial guess is given by Equations 4.4.6 and 4.4.8, as discussed in Section 3.3.
- The harmonic function for the previous configuration<sup>30</sup> can also be used as part of formulating the initial guess for the new configuration. The function available during convergence towards the harmonic function, or if available, the harmonic function for the old configuration, can be used to formulate the initial guess for the new configuration. Let  $U_{old}^{(k)}$  be the  $k$ ’th iteration of the harmonic function computation using the old configuration. If  $U_{old}^{(k)}$  is equal to  $U_{old}^{(\infty)}$ , then convergence had been achieved for the old configuration.  $U_{new}^{(0)}$  is the potential function used as an initial guess to the harmonic function for the new configuration. There are four ways in which the configuration of obstacles and goals may change (see Figure 4.8) and therefore four ways of formulating the initial guess for the recomputation of the harmonic function. The methods depend on the type of changes to the obstacle and goal configuration and can be stated as follows:

- (i) In the first case, a new obstacle object model  $\beta_i$  is added to the configuration space. This changes the free space  $C_{free}$  on which the potential function  $U$  is defined:

$$(4.4.26) \quad C_{free(new)} = C_{free(old)} - \beta_i$$

Let  $U_{rep_i}$  be a local potential function (i.e., one of the functions specified in Equation 4.4.5) associated with the obstacle object model  $\beta_i$ . The initial guess to the solution of the changed configuration is given by the following:

$$(4.4.27) \quad U_{new}^{(0)} = \max(U_{old}^{(k)}, U_{rep_i})$$

<sup>30</sup>The free space will change by either the addition or subtraction of grid elements. The values at the unchanging grid elements can be used to formulate the initial guess for the new configuration.

- (ii) In the second case, an existing obstacle object model  $\beta_i$  is removed from the configuration space. This has the effect of increasing the extent of the free space  $C_{free}$ :

$$(4.4.28) \quad C_{free(new)} = C_{free(old)} + \beta_i$$

For this situation, the initial guess to the solution of the changed configuration is given as follows:

$$(4.4.29) \quad U_{new}^{(0)} = U_{old}^{(k)}$$

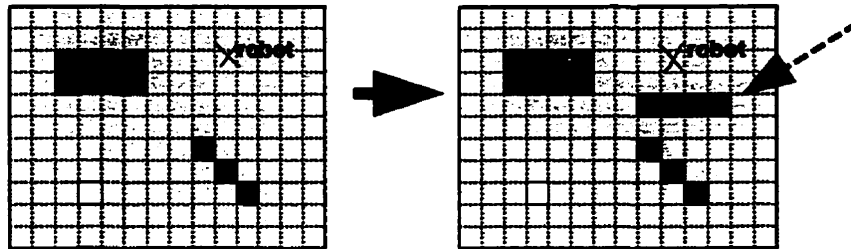
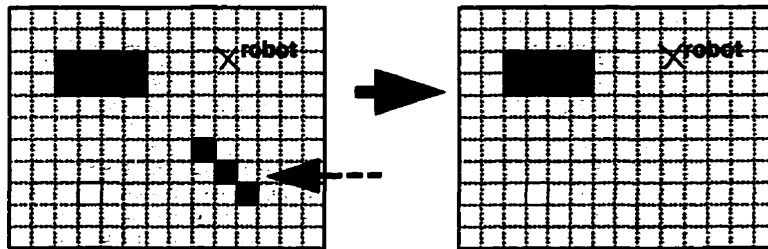
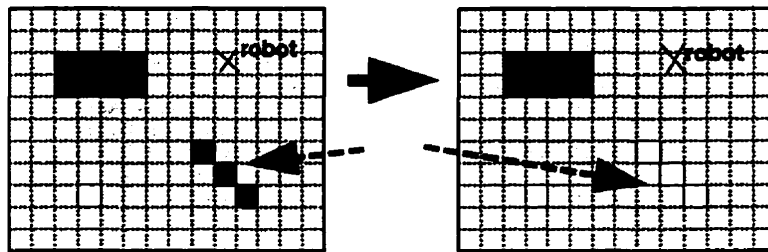
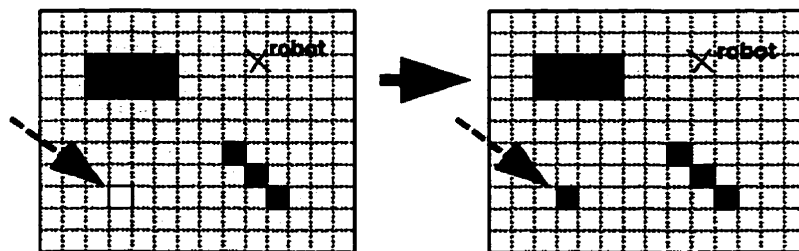
- (iii) In the third case, an obstacle object model  $\beta_i$  in the configuration space is changed to a goal object model  $\beta_{i*}$ . This does not affect the free space  $C_{free}$ . Let  $U_{att_i}$  be a local potential function (i.e., one of the functions in Equation 4.4.5) associated with the goal object model  $\beta_{i*}$ . The initial guess to the solution of the changed configuration is given by the following:

$$(4.4.30) \quad U_{new}^{(0)} = \min(U_{old}^{(k)}, U_{att_i})$$

- (iv) In the fourth case, a goal object model  $\beta_{i*}$  is changed to an obstacle object model  $\beta_i$ . As in the third case, the free space  $C_{free}$  remains unchanged. The initial guess to the solution of the changed configuration is given by the following:

$$(4.4.31) \quad U_{new}^{(0)} = \max(U_{old}^{(k)}, U_{rep_i})$$

A good initial guess to the computation of a harmonic function reduces the number of iterations required for convergence. Using an initial guess, as well as the *method of relaxation* (outlined in Section 4.1.1), is the extent of what can be done to reduce the number of iterations. The time for a single iteration is an implementations issue and dealt with in Chapter 6. At the same time the harmonic function is being computed, a concurrent agent (i.e., another process) performs steepest gradient descent on the harmonic function in order to generate a local trajectory. This permits concurrent computation and execution of the path.

**i) ADDING A NEW OBSTACLE OBJECT MODEL****ii) REMOVING AN EXISTING OBSTACLE OBJECT MODEL****iii) CHANGING AN OBJECT MODEL FROM AN OBSTACLE TO A GOAL****iv) CHANGING AN OBJECT MODEL FROM A GOAL TO AN OBSTACLE**

**FIGURE 4.8.** Types of Changes to the Obstacle and Goal Configuration. The grey pixels signify the free space, whereas the black pixels represent the obstacle models and the white pixels represent the goal models. The configuration space is the collection of white, black and grey pixels. The figure illustrates the four different possible changes to the configuration space. All other potential changes are a combination of the above cases.

## 5. Trajectory Generation Using Harmonic Functions

### 5.1. Local Trajectory Generation

*Harmonic functions* have the desirable property of no local minima (Connolly & Gruppen, 1993; Tarassenko & Blake, 1991). If at least one path exists to a known destination, steepest gradient descent on this function is guaranteed to find it (Doyle & Snell, 1984). This is advantageous for path execution because steepest gradient descent is a simple and blind search strategy that is computationally inexpensive (i.e., no backtracking).

The derivative of the harmonic function  $U$  at every free space  $C_{free}$  point defines a field of velocity vectors  $\vec{V}(e)$ :

$$(4.4.32) \quad \vec{V}(e) = -\nabla \vec{U}(e)$$

where  $\nabla \vec{U}(e)$  denotes the gradient vector of  $U$  at  $(e)$ :

$$(4.4.33) \quad \nabla \vec{U} = \begin{pmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{pmatrix}$$

The velocity vector is the value of  $\vec{V}(e)$  at the current estimate of the robot's position (i.e.,  $\vec{V}(x_r, y_r)$ ). The field of velocities permits robot localization (i.e., the correction of the robot's position) to be independent of the trajectory generation. If the robot's position is updated (i.e., corrected) to  $(x_r', y_r')$ , then the trajectory will be selected based on the value  $\vec{V}$  at  $(x_r', y_r')$  (i.e.,  $= \vec{V}(x_r', y_r')$ ). The velocity field defines all possible trajectories from every location in the free space.

The velocity vector  $\vec{V}(e)$  has two components: (1) an orientation, and (2) a magnitude. The orientation component defines the direction of the next robot movement.

A simple and fast algorithm is used to compute the local orientation component of the velocity (see Figure 4.9). It is calculated by selecting the maximum negative gradient in an eight-neighbourhood<sup>31</sup> around the current position of the robot. The result will be an orientation value that is divisible by 45 degrees. In order to obtain a smoother trajectory, an approximation to the continuous gradient is obtained by fitting a parabola to the three points defined by the maximum negative gradient direction and its two neighbours (i.e., in the eight-neighbourhood operator centered at the current position of the robot). The local minimum of the fitted parabola defines the orientation of the local trajectory.

It would be desirable to have the magnitude of the velocity vector  $|\vec{V}(e)|$  define the speed of the robot. The field of velocities  $\vec{V}(e)$  generated from a harmonic function is such

<sup>31</sup>Recall that the potential function is defined on a discretized grid.

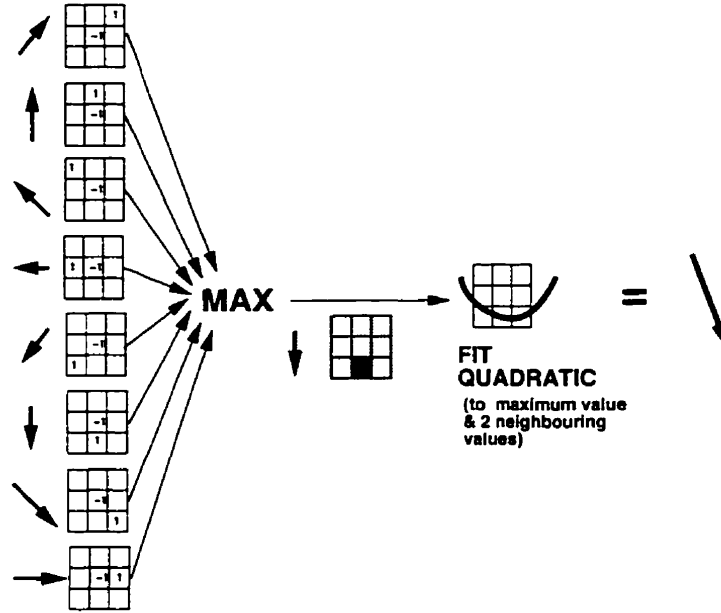


FIGURE 4.9. Quick Calculation of Trajectory. The local trajectory is derived by selecting the maximum negative gradient in an eight-neighbourhood around the current position of the robot in the discretized potential function. In order to obtain smooth trajectories, an interpolated value is computed by fitting a parabola to the three points defined by the maximum negative gradient direction and its two neighbours.

that  $|\vec{V}(e)|$  is a larger value near goal locations when compared to other locations in the free space. A possible strategy for velocity control would be to make the robot's speed inversely proportional to the gradient's magnitude  $|\vec{V}(e)|$ :

$$(4.4.34) \quad |\vec{V}(e)| \propto -\frac{1}{|\nabla \tilde{U}(e)|}$$

This would result in the robot de-accelerating as it progressed towards the goal position. However, this is not an appropriate strategy to take because the robot would start at a very high speed (i.e., jump start) and deaccelerate as it approached the goal position. In the more general case, the determination of the velocity should depend on a predefined velocity profile from the start to goal position. The profile should consist of a combination of the following velocity states: (1) acceleration, (2) constant velocity, and (3) de-acceleration. A maximum velocity for the actuator sets an absolute limit. Depending on the arrangement of the velocity states from the start to goal position, the velocity will either be directly or inversely proportional to the harmonic function gradient at a particular location. Unfortunately, the proportional factor is not constant, since it depends on the positioning of the object and goal models. Thus, it is not clear on how to use the potential function gradient information



in order to control speed, but the gradient vector  $\vec{V}(x_r, y_r)$  can definitely be used to specify the trajectory.

The optimality of the generated trajectory is established by noting that there is a direct correspondence between harmonic functions and probability functions for a random walk on a Markov<sup>32</sup> chain (Doyle & Snell, 1984). The probabilistic interpretation of each point in the harmonic function is a *hitting probability* (Connolly, 1994) (i.e., the probability of hitting an obstacle before arriving at the goal). The steepest gradient-descent path in a harmonic function is optimal in terms of the described probabilistic interpretation. For Dirichlet boundary conditions, steepest gradient descent produces a path where the hitting probability is continually and maximally being reduced. The path is smooth<sup>33</sup> and optimal in the sense of being the shortest path subject to a penalty for proximity to obstacles (i.e., the path wants to move away from obstacles). In the Neumann case, the derivative of the potential function is zero at obstacle boundaries, and consequently the path is not subject to the obstacle proximity penalty.

---

<sup>32</sup>A Markov process is a random process whose transition probabilities at the current time do not depend on prior transitions. A finite Markov chain is a special type of change process that moves around the set of states  $S = \{s_1, s_2, \dots, s_r\}$  in a Markovian fashion. When the process is in state  $s_i$ , it moves with probability  $P_{ij}$  to state  $s_j$ . States that are once entered and cannot be left are called *traps* or *absorbing-states*. A Markov chain is called absorbing if it has at least one absorbing state and if, from any state, it is possible (not necessarily in one step) to reach at least one absorbing state. The states of an absorbing chain that are not traps are called *non-absorbing*.

<sup>33</sup>Almost all paths on a harmonic function are infinitely differentiable along their length (Sabersky *et al.*, 1971). An exception is a saddle point which will still be piecewise differentiable. The path produced from steepest gradient descent will always escape a saddle point and is therefore infinitely differentiable along its length.

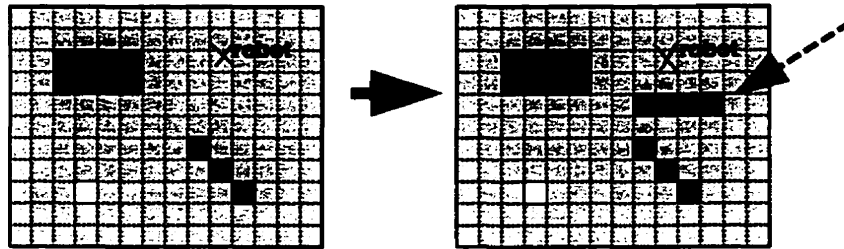
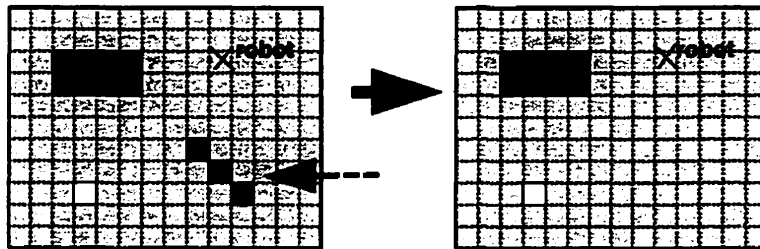
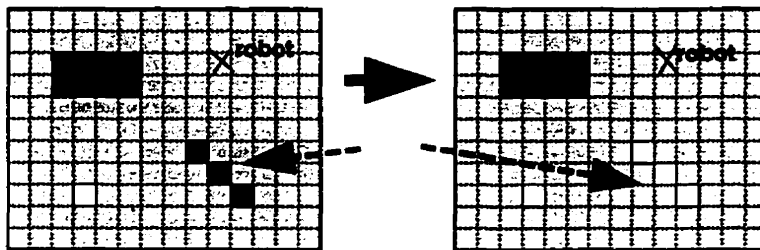
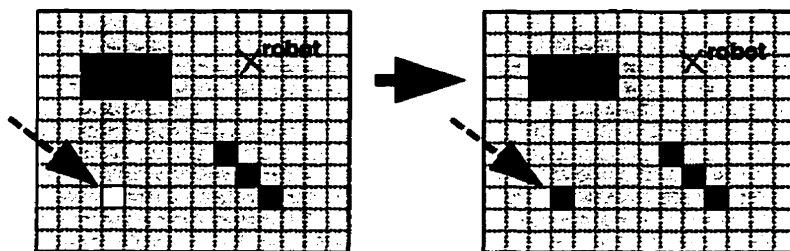
**i) ADDING A NEW OBSTACLE OBJECT MODEL****ii) REMOVING AN EXISTING OBSTACLE OBJECT MODEL****iii) CHANGING AN OBJECT MODEL FROM AN OBSTACLE TO A GOAL****iv) CHANGING AN OBJECT MODEL FROM A GOAL TO AN OBSTACLE**

FIGURE 4.8. Types of Changes to the Obstacle and Goal Configuration. The grey pixels signify the free space, whereas the black pixels represent the obstacle models and the white pixels represent the goal models. The configuration space is the collection of white, black and grey pixels. The figure illustrates the four different possible changes to the configuration space. All other potential changes are a combination of the above cases.

## 5. Trajectory Generation Using Harmonic Functions

*Harmonic functions* have the desirable property of no local minima (Connolly & Grupen, 1993; Tarassenko & Blake, 1991). If at least one path exists to a known destination, steepest gradient descent on this function is guaranteed to find it (Doyle & Snell, 1984). This is advantageous for path execution because steepest gradient descent is a simple and blind search strategy that is computationally inexpensive (i.e., no backtracking).

The derivative of the harmonic function  $U$  at every free space  $C_{free}$  point defines a field of velocity vectors  $\vec{V}(e)$ :

$$(4.4.32) \quad \vec{V}(e) = -\nabla \vec{U}(e)$$

where  $\nabla \vec{U}(e)$  denotes the gradient vector of  $U$  at  $(e)$ :

$$(4.4.33) \quad \nabla \vec{U} = \begin{pmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{pmatrix}$$

The velocity vector is the value of  $\vec{V}(e)$  at the current estimate of the robot's position (i.e.,  $\vec{V}(x_r, y_r)$ ). The field of velocities permits robot localization (i.e., the correction of the robot's position) to be independent of the trajectory generation. If the robot's position is updated (i.e., corrected) to  $(x_r', y_r')$ , then the trajectory will be selected based on the value  $\vec{V}$  at  $(x_r', y_r')$  (i.e.,  $= \vec{V}(x_r', y_r')$ ). The velocity field defines all possible trajectories from every location in the free space.

The velocity vector  $\vec{V}(e)$  has two components: (1) an orientation, and (2) a magnitude. The orientation component defines the direction of the next robot movement.

A simple and fast algorithm is used to compute the local orientation component of the velocity (see Figure 4.9). It is calculated by selecting the maximum negative gradient in an eight-neighbourhood<sup>31</sup> around the current position of the robot. The result will be an orientation value that is divisible by 45 degrees. In order to obtain a smoother trajectory, an approximation to the continuous gradient is obtained by fitting a parabola to the three points defined by the maximum negative gradient direction and its two neighbours (i.e., in the eight-neighbourhood operator centered at the current position of the robot). The local minimum of the fitted parabola defines the orientation of the local trajectory.

It would be desirable to have the magnitude of the velocity vector  $|\vec{V}(e)|$  define the speed of the robot. The field of velocities  $\vec{V}(e)$  generated from a harmonic function is such that  $|\vec{V}(e)|$  is a larger value near goal locations when compared to other locations in the free

<sup>31</sup> Recall that the potential function is defined on a discretized grid.

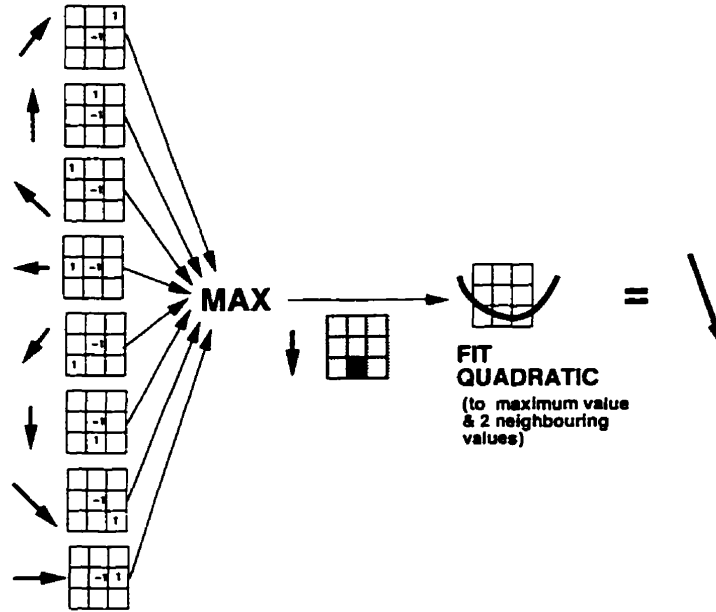


FIGURE 4.9. **Quick Calculation of Trajectory.** The local trajectory is derived by selecting the maximum negative gradient in an eight-neighbourhood around the current position of the robot in the discretized potential function. In order to obtain smooth trajectories, an interpolated value is computed by fitting a parabola to the three points defined by the maximum negative gradient direction and its two neighbours.

space. A possible strategy for velocity control would be to make the robot's speed inversely proportional to the gradient's magnitude  $|\vec{V}(e)|$ :

$$(4.4.34) \quad |\vec{V}(e)| \propto -\frac{1}{|\nabla \bar{U}(e)|}$$

This would result in the robot de-accelerating as it progressed towards the goal position. However, this is not an appropriate strategy to take because the robot would start at a very high speed (i.e., jump start) and deaccelerate as it approached the goal position. In the more general case, the determination of the velocity should depend on a predefined velocity profile from the start to goal position. The profile should consist of a combination of the following velocity states: (1) acceleration, (2) constant velocity, and (3) de-acceleration. A maximum velocity for the actuator sets an absolute limit. Depending on the arrangement of the velocity states from the start to goal position, the velocity will either be directly or inversely proportional to the harmonic function gradient at a particular location. Unfortunately, the proportional factor is not constant, since it depends on the positioning of the object and goal models. Thus, it is not clear on how to use the potential function gradient information in order to control speed, but the gradient vector  $\vec{V}(x_r, y_r)$  can definitely be used to specify the trajectory.

The optimality of the generated trajectory is established by noting that there is a direct correspondence between harmonic functions and probability functions for a random walk on a Markov<sup>32</sup> chain (Doyle & Snell, 1984). The probabilistic interpretation of each point in the harmonic function is a *hitting probability* (Connolly, 1994) (i.e., the probability of hitting an obstacle before arriving at the goal). The steepest gradient-descent path in a harmonic function is optimal in terms of the described probabilistic interpretation. For Dirichlet boundary conditions, steepest gradient descent produces a path where the hitting probability is continually and maximally being reduced. The path is smooth<sup>33</sup> and optimal in the sense of being the shortest path subject to a penalty for proximity to obstacles (i.e., the path wants to move away from obstacles). In the Neumann case, the derivative of the potential function is zero at obstacle boundaries, and consequently the path is not subject to the obstacle proximity penalty.

---

<sup>32</sup>A Markov process is a random process whose transition probabilities at the current time do not depend on prior transitions. A finite Markov chain is a special type of change process that moves around the set of states  $S = \{s_1, s_2, \dots, s_r\}$  in a Markovian fashion. When the process is in state  $s_i$ , it moves with probability  $P_{ij}$  to state  $s_j$ . States that are once entered and cannot be left are called *traps* or *absorbing-states*. A Markov chain is called absorbing if it has at least one absorbing state and if, from any state, it is possible (not necessarily in one step) to reach at least one absorbing state. The states of an absorbing chain that are not traps are called *non-absorbing*.

<sup>33</sup>Almost all paths on a harmonic function are infinitely differentiable along their length (Sabersky *et al.*, 1971). An exception is a saddle point which will still be piecewise differentiable. The path produced from steepest gradient descent will always escape a saddle point and is therefore infinitely differentiable along its length.

## 6. Guaranteeing Proper Control

There is no guarantee on the correctness of the trajectory commands before the computation has converged to the solution for the current configuration. In order to address the correctness of the trajectory, a type of “*anytime algorithm*” (Mouaddib & Zilberstein, 1995) is proposed. *Anytime algorithms* are algorithms whose quality of results improves as computation time increases. There are three metrics to measure quality (Mouaddib & Zilberstein, 1995): (1) *certainty* in result correctness; (2) *accuracy* in result correctness; and (3) *specificity* in the level of detail of the result (i.e., sampling). The proposed method improves along all three metrics as time increases.

The iterative computation of the harmonic function for a given configuration takes a fixed amount of time before convergence is achieved. Polling the gradient before convergence to the harmonic function will result in potentially erroneous trajectories. Thus, it is desirable to have the computation converge to the harmonic function almost instantaneously. The computation time is exponentially proportional to the number of discrete grid elements. It would be desirable to reduce the number of grid elements (i.e., make each grid element represent a larger spatial area), but the fewer their number, the coarser the path. A reasonable tradeoff is to simultaneously compute the harmonic function for different grid resolutions<sup>34</sup> (see Figure 4.10), and use the coarse grid result initially, gradually progressing to using the finer grid results as their associated computations converge<sup>35</sup>.

Let the different grid levels be  $\{g_1, g_2, \dots, g_n\}$ , where  $g_1$  is the coarsest (i.e., lowest) resolution grid and  $g_n$  is the finest (i.e., highest) resolution grid. Given equal computational resources,  $g_i$  will converge before  $g_j$  if  $j > i$ . All grids are given the same initial obstacle and goal configuration and are all updated with newly acquired sensor information as it becomes available. The harmonic functions are all computed iteratively and concurrently for each grid resolution. The steepest descent gradient is taken at the current estimated position of the robot for each grid computation:  $\{\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n\}$ . If grid  $g_i$  has converged

<sup>34</sup>From a theoretical point of view, such a strategy is feasible. Each harmonic function is computed at a different resolution and each one is a suitable candidate for the generation of the trajectory. However, the harmonic function with more grid elements (i.e., finer resolution) will produce a smoother trajectory. From a practical point of view, there is a resource limitation on the number of levels in the hierarchy. Five processors were usually used to compute a single harmonic function (see Chapter 7). Therefore, experimentation with SPOTT was confined to using only two levels in the hierarchy.

<sup>35</sup>Ideally, to save on computational resources, it would be desirable to compress the hierarchy into a single level with an unequal grid sampling. Large open spaces would be represented by a few grid elements, while small tight spaces would be represented by a more dense sampling. The local sampling might have to dynamically change depending on what is sensed from the environment. The computational overhead for managing a data structure capable of representing all the hierarchies within one level should not slow down the reaction of the system (i.e., for obstacle avoidance) to newly sensed features. Representing the hierarchy in a single level is not addressed by this thesis and is left as a future research topic.

to the harmonic function for the current configuration, and there are no grids  $g_j$ , where  $j > i$ , that have also converged, then  $\vec{v}_i$  is used as the local trajectory control. It is assumed that the computation time for convergence at the coarsest grid resolution is faster than the rate of change in the environment. Therefore, if there are no environmental changes sensed, eventually grid  $g_n$  (i.e., the finest grid) will converge and its trajectory vector  $\vec{v}_n$  will be used to specify the local trajectory for the robot. For normal operation, the trajectory vectors are used in the order of ascending grid resolution  $\{\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n\}$ . This progression switches from grid  $g_i$  to grid  $g_{i+1}$  (i.e., control from  $\vec{v}_i$  to  $\vec{v}_{i+1}$ ) when grid  $g_{i+1}$  has converged to its solution. The progression is restarted (i.e., sometimes before grid  $g_n$ 's vector  $\vec{v}_n$  has had a chance to be used for control) when the local map is updated (e.g., newly sensed data arrives, or a new goal is set). At this time,  $\vec{v}_1$  will again be used for control, and control will again switch in ascending order until  $\vec{v}_n$  is reached or newly sensed data arrives and control is switched back to  $\vec{v}_1$ . The switching between levels depends on determining if a particular grid resolution  $g_i$  has converged to its harmonic function.

Convergence can be found by measuring the change in value of the harmonic function grid elements between successive iteration steps. It is achieved when this value is a minimum (i.e., approximately zero). However, convergence monitoring requires additional computational resources, which will slow down the computation of the harmonic function. An alternative approach is to compare all the trajectory vectors  $\{\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n\}$  in ascending order, assuming that  $\vec{v}_1$ , the direction computed at the coarsest resolution, is correct. If  $\vec{v}_2$  is approximately<sup>36</sup> equal to  $\vec{v}_1$ , then  $\vec{v}_2$  is used, otherwise  $\vec{v}_1$  is used. This comparison continues between successive grid resolution levels until it fails or  $\vec{v}_n$  is reached.

Section 5 discussed the optimality of the trajectory produced by performing steepest gradient descent on the harmonic function by the local path planner. This section addressed the ability to guarantee proper response (i.e., control) in a real-time operational scenario. The potential field (i.e., local path planner) only performs path planning within a local window of the map (i.e., the world known to SPOTT). The next section addresses the issue of integrating a topologically-based global path planner with the presented local path planner in order to perform path planning in a larger scale environment.

---

<sup>36</sup>In SPOTT's implementation, "approximately equal" means being within  $\pm 22.5^\circ$  of the compared value. If  $\vec{v}_i$  is equal to  $\vec{v}_{i+1} \pm 22.5^\circ$ , then  $\vec{v}_{i+1}$  is used for control. Otherwise,  $\vec{v}_i$  is continued to be used for control.

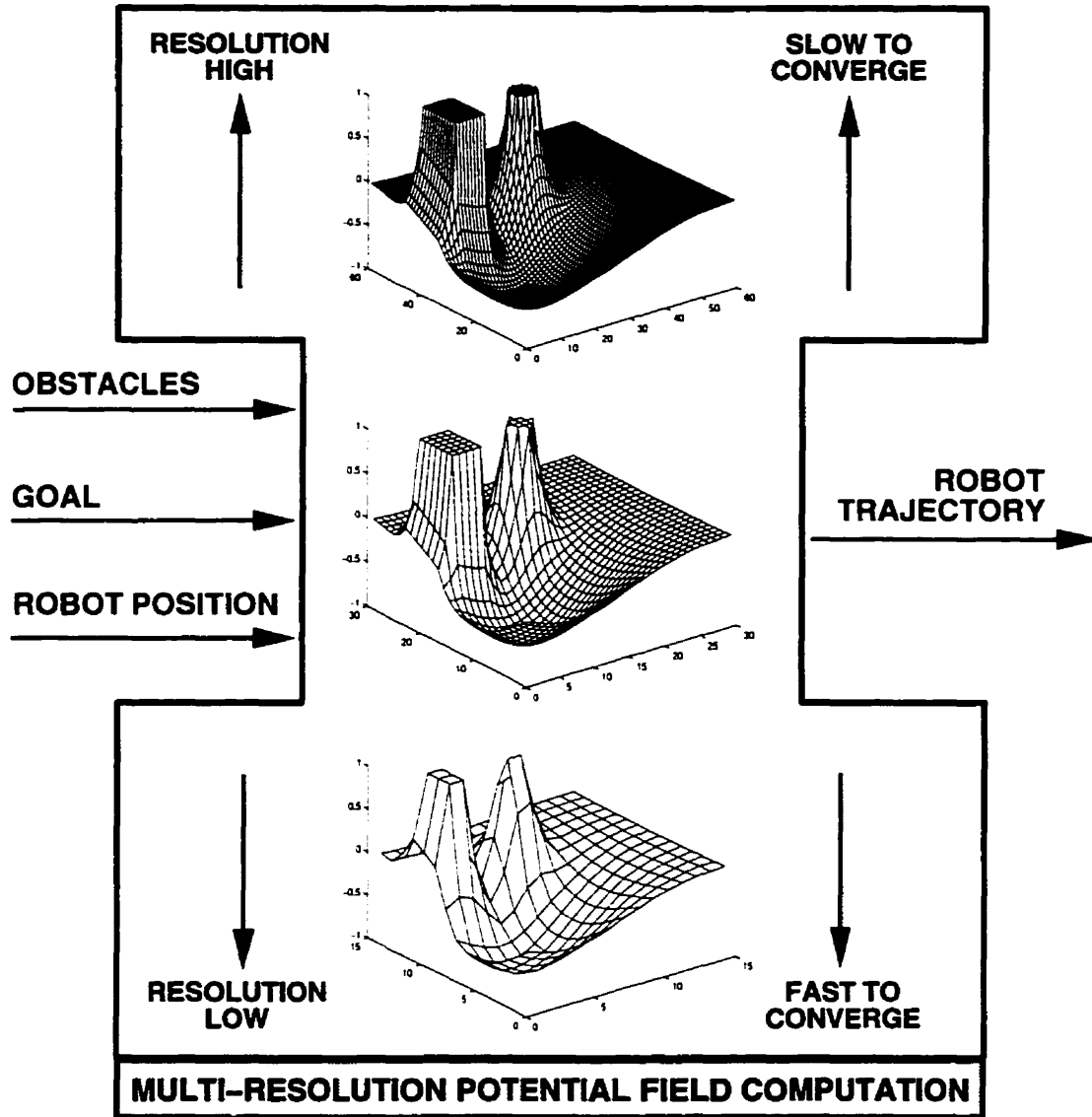


FIGURE 4.10. Multi-Resolution Potential Fields. The trajectory of the robot is continuously available by taking the gradient at the finest resolution that has converged to its solution (i.e., harmonic function) at a particular time instance.



## 7. Why is the Potential Field a Local Path Planner?

The potential field is not computable over a large environmental space due to the enormous computational costs<sup>37</sup> associated with a large number of grid elements, and the rapidly decaying nature of the harmonic function. The environment used in the experiments for SPOTT is an office and laboratory space which is 6500 cm by 3990 cm. The largest size an individual grid element can be is 30<sup>2</sup> cm, which was determined by using three grid elements to occupy the space between the two closest features in the environment. In an office and laboratory space, the narrowest navigational regions are usually hallways and doorways<sup>38</sup>. Reasonable convergence times (i.e., less than two seconds) can be obtained for grids which are 35 by 35 (see Chapter 7). Thus, the maximum size a potential field can cover is approximately 10 m by 10 m.

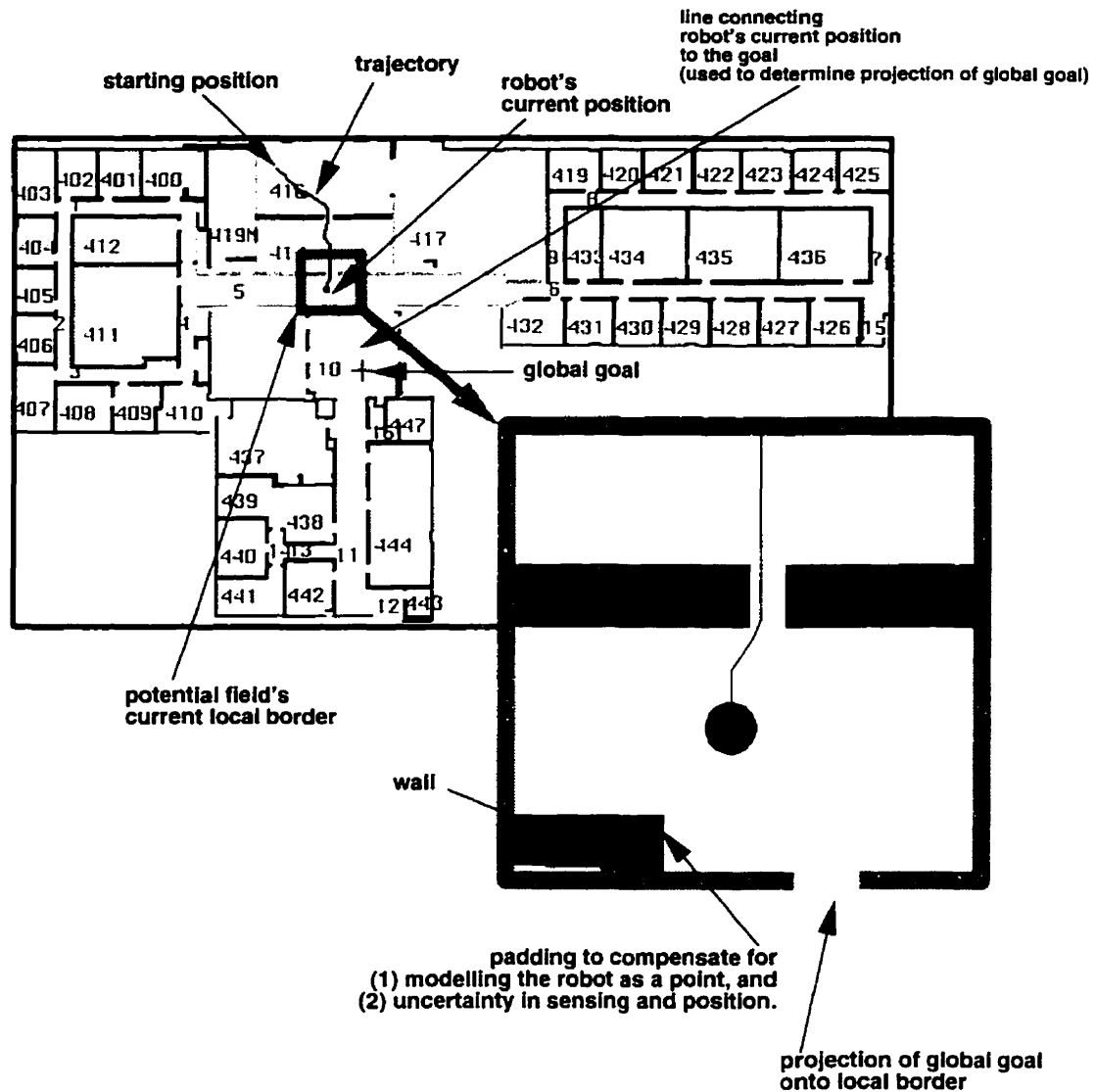
In addition, the extent of the potential field is further constrained due to the rapid decay of the harmonic function with Dirichlet boundary conditions in narrow corridors. On a thirty-two bit computer, the harmonic function can only be accurately represented when the length-to-width ratio for a narrow corridor is less than 7.1 to 1 (see Appendix A).

Therefore the potential field can only be used as a local path planner and it must slide around to keep the robot within its bounds. When the robot reaches the frontier of the local region, the bounds are shifted. The size of the potential function is fixed and its new bounds centre the robot (see Figure 4.11). A global goal outside the extent of the potential function's boundaries projects onto the border using information provided by the global path planner (see Section 9)

---

<sup>37</sup>Computation time is exponentially proportional to the number of grid elements.

<sup>38</sup>This is true when only considering the features present in an architectural CAD map of the environment. There may be other regions where desks and chairs are closely positioned together, but this space is not traversed by the robot in the experiments discussed in this thesis.



**FIGURE 4.11. Sliding Local Path Planner.** The potential function's boundary slides when the robot approaches the current boundary. The new boundary is defined so that the current position of the robot is at its centre. The global path planner provides the information which projects the global goal, which is outside the local extent, onto the current local boundary (see Section 9). The shading around the wall features corresponds to their padding in order to compensate for modelling the robot as a point, as well as the sensing and positioning uncertainty.

## 8. Global Path Planning

An architectural CAD map (see Figure 4.12) consists of a collection of line segments defining internal walls and the external construction frame. It is assumed that abstract symbolic entities in the map (e.g., rooms and hallways) are labelled. This abstract representation is used by the global path planner to provide the local path planner with the necessary information to guide it towards a goal located outside its current local extent.

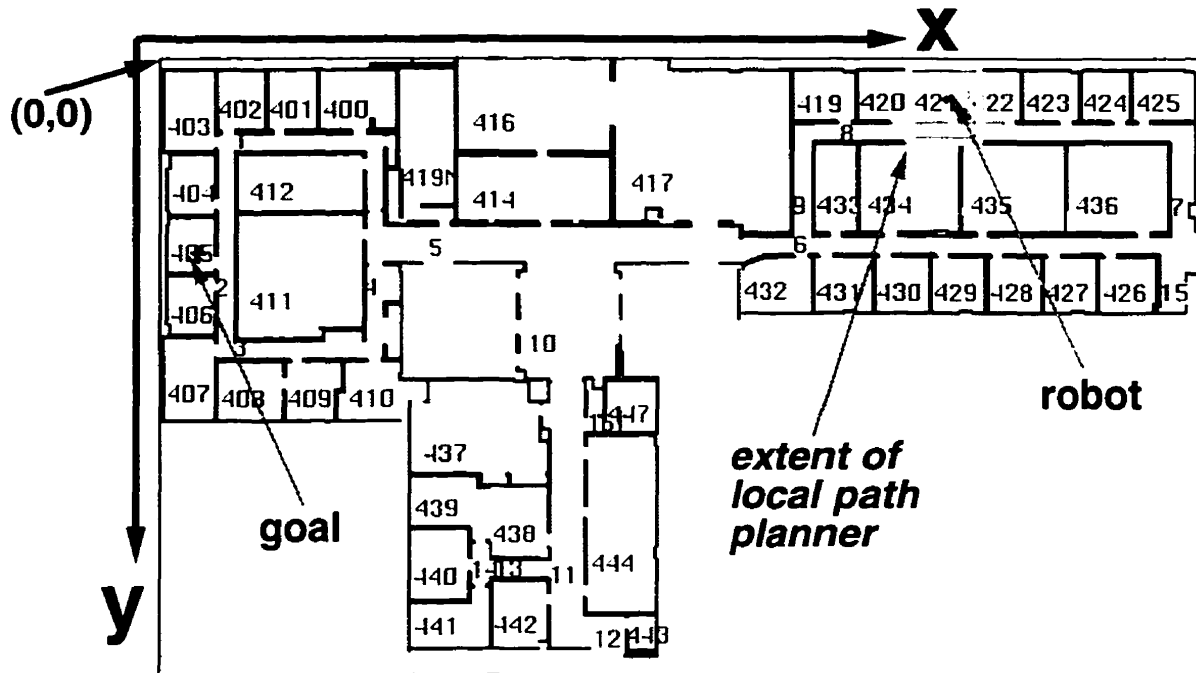


FIGURE 4.12. CAD Map of CIM. The features are all line segments representing permanent walls. The extent of the local path planner at a particular instance of time is shown.

The abstract graph structure (see Figure 4.13) is defined by  $G(N, E)$ .  $N$  is a set of nodes  $\{n_1, n_2, \dots\}$  and  $E$  is a set of edges  $\{e_1, e_2, \dots\}$ . Each edge  $e$  connects the elements of an unordered distinct pair of nodes  $\{n_i, n_j\}$ . Every node represents a room or a rectangular portion of hallway. The summation of the rectangular portions of hallways combine to form the entire hallway network. An edge represents an access way between nodes. An access way can either be a doorway (i.e., open or closed) or a virtual plane<sup>39</sup>, which projects to a line in 2D. A doorway's state (i.e., open or closed) may change depending on perceptual information which is continually being received from the sensors<sup>40</sup>. The search for a path

<sup>39</sup>This is the case for two adjoining rectangular hallway portions where there is no doorway between them. The boundary is artificially created to satisfy the self-imposed requirement that nodes represent rectangular regions.

<sup>40</sup>Currently, QUADRIIS (i.e., laser rangefinder) is used to recognize door states (i.e., open or closed).

in the graph space translates to a graph search from a start  $n_r$  to a destination node  $n_g$ . A *path* is a sequence of edges  $e_1, e_2, \dots$  such that:

- (i)  $e_i$  and  $e_{i+1}$  share a common endpoint;
- (ii)  $e_i$  is not a self-loop; and
- (iii) if  $e_i$  is not the first or last edge,  $e_i$  shares one of its endpoints with  $e_{i-1}$  and the other with  $e_{i+1}$ .

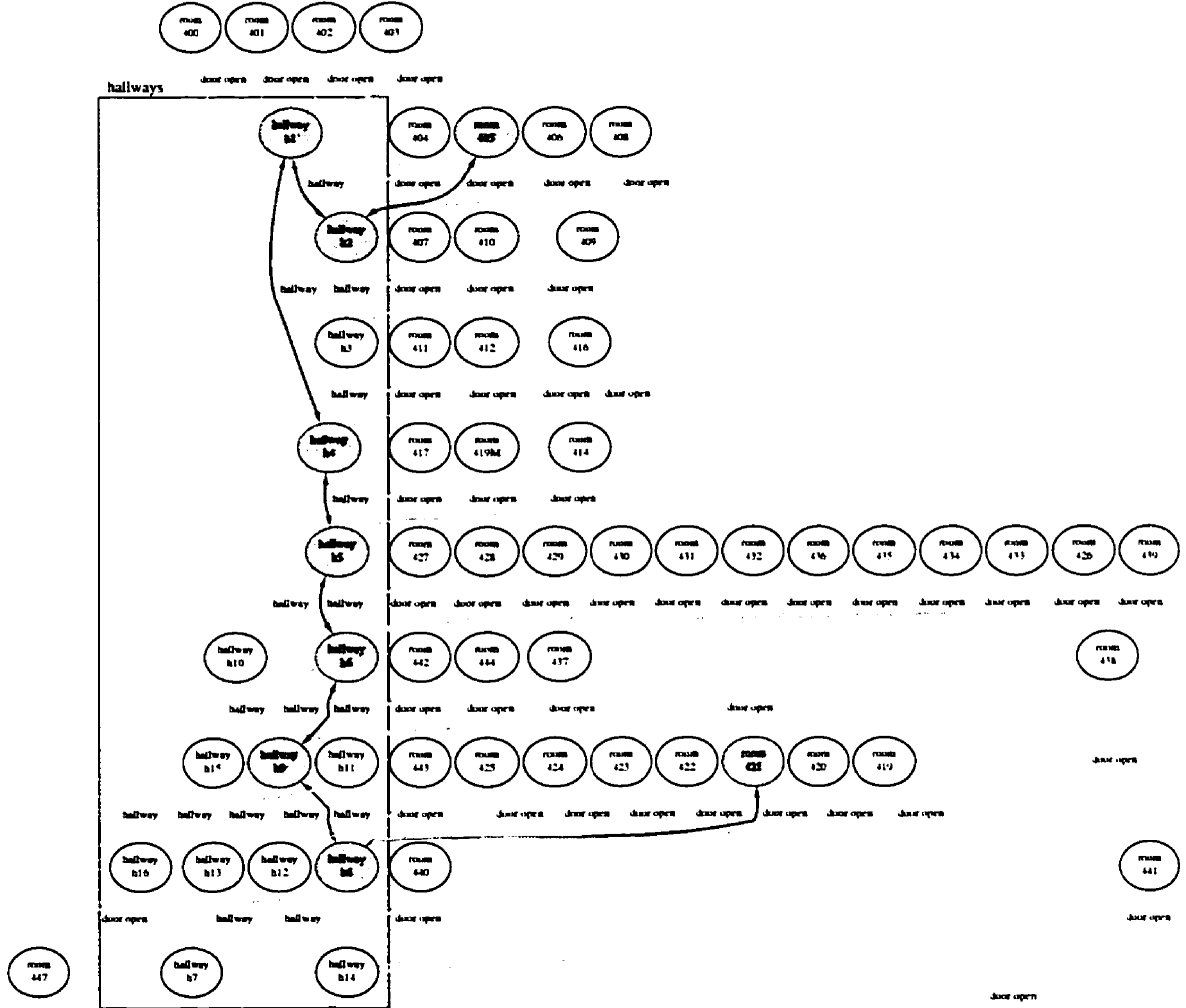


FIGURE 4.13. Abstract Graph of CAD Map. This is the abstract graph corresponding to the CAD map in Figure 4.12. The nodes represent rooms or hallway portions. The edges represent access ways (i.e., doors or virtual planes) which are line features in the CAD map. The highlighted edges represent the path produced by Dijkstra's algorithm using the robot and goal locations shown in Figure 4.12. The robot starts in room 421 and its destination is room 405.

A path  $\{n_r, n_i, n_{i+1}, \dots, n_g\}$  is represented by the collection of nodes visited by traversing the edges from a starting position  $n_r$  to a goal position  $n_g$ . Dijkstra's algorithm (Aho *et al.*, 1983) is executed on the graph in order to find the shortest path from the starting

node to the goal node. The path is computed when the task (i.e., go from  $n_r$  to  $n_g$ ) is specified and recomputed only when an edge changes state (e.g., from an opened door to a closed door). Initially, it is assumed that all doors are open until verified or refuted by sensor data.

In order to obtain a shortest path from the graph structure, a notion of distance with respect to the abstract graph is defined. A Cartesian reference is superimposed on the CAD map, such that the top left hand corner is the origin. The positive x axis is directed to the right while the positive y axis is oriented downwards (see Figure 4.12). All the rooms in the environment are rectangular and are represented by two sets of endpoints, one representing the top left hand corner and the other, the bottom right hand corner:  $n_{room} = ((x_{rm1}, y_{rm1}), (x_{rm2}, y_{rm2}))$ . Access ways are represented by a line segment:  $e_i = ((x_{a1}, y_{a1}), (x_{a2}, y_{a2}))$ . The distance from the start node to the next node is the distance from the robot's starting position to the access way of the next node. The distance for intermediate nodes on the path is the distance from the access way by which the node was entered to the access way that connects it to the next node. The distance from the second-last node on the list to the goal is the distance from the access way from which the second-last node was entered to the position of the goal via the access way connecting the second-last and goal nodes (see Figure 4.14).

Architectural CAD maps can be a valuable piece of a priori knowledge for navigating a robot in indoor environments. The CAD map only contains the permanent structures in the environment (i.e., walls). The robot still has to sense the environment to discover objects not contained in the CAD map and to verify or refute<sup>41</sup> the existence and location of the CAD map features (e.g., walls). A local window into the CAD map is used as an initial map for the local path planner. The graphical abstraction of the CAD map (i.e., nodes and edges) is used to guide the robot towards the boundary of the local region in which local path planning is done in order to satisfy the global goal.

---

<sup>41</sup>Refutation is currently not done by SPOTT. However, SPOTT does verify the existence and label (e.g., door, wall) of existing features. All CAD features are initially labelled as walls. Both sonar and QUADRIS data are used for mapping the environment, but only QUADRIS is used to recognize objects (e.g., doors, walls).

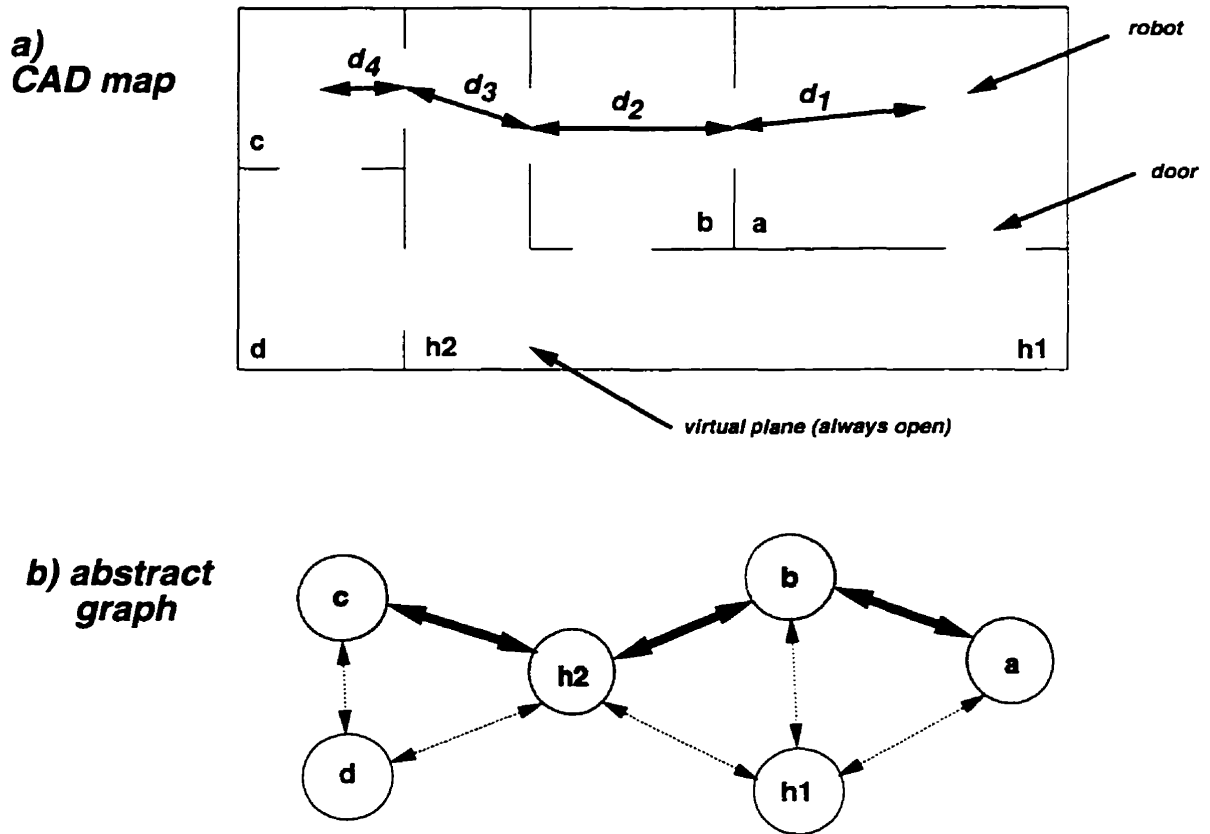


FIGURE 4.14. Calculating Distances for Global Path Planning. (a) shows a sample CAD map with a path going from room *a* to room *c*. The corresponding abstract graph is shown in (b), where the nodes to be visited are shaded and the traversed edges along the path are in bold type. The distance from the start node to the next node is the distance from the robot's starting position to the access way of the next node (i.e.,  $d_1$ ). The distance for intermediate nodes on the path is the distance from the access way by which the node was entered to the access way that connects it to the next node (i.e.,  $d_2$ ). The distance from the second-last node on the list to the goal is the distance from the access way from which the second-last node was entered to the position of the goal via the access way connecting the second-last and the goal node (i.e.,  $d_3 + d_4$ ).

## 9. Local and Global Path Planner Interaction

The global path planner, by using an abstraction of the CAD map, is able to provide global goal information to the local path planner (i.e., potential field). This occurs when the bounds of the potential field change: as the robot approaches the border of the potential field, the potential field is re-anchored with the robot position in the centre, and the global goal is projected onto the potential field's new border. If the global goal is within the potential field's extent, then there is no need to project the global goal onto the border. If no CAD map is available, then the location of the global goal is projected onto the border by joining the global goal to the robot position, and intersecting this line with the potential field's border. On the other hand, if a CAD map is available and the spatial location of the goal is outside the extent of the potential field, the goal is projected onto the border of the potential field at a location determined by an algorithm (see Section 9.3) which uses the global path produced by the global path planner.

### 9.1. Four Typical Scenarios

The four typical scenarios under which global goal information is used (i.e., if it is necessary) by the local path planner for determining the projection of the global goal onto its boundary are as follows:

- (i) In the first case, the position of the robot and the goal are within the local extent of the potential field, and the goal is reachable (i.e., there is a path to the goal that stays within the local confines of the potential field) from the robot's current position (see Figure 4.15).
- (ii) In the second case, the position of the robot and goal are within the local extent of the potential field, but the goal is unreachable (i.e., there is no path to the goal that stays in the local confines of the potential field) from the current robot's position (see Figure 4.16). The cause of this is either that the access way into the room, where the goal is located, is not contained within the local extent, or the access way is blocked (i.e., closed door). The global path is used to determine the projection of the goal onto the proper location on the boundary of the potential field (see Section 9.3).
- (iii) In the third case, the robot and the goal are located in the same node (e.g., room), but the goal is not within the local extent of the potential field (see Figure 4.17). The intersection of a line joining the robot and the goal with the potential field's border is used to determine the location of the projection of the goal onto the potential field's borders (see Figure 4.20).

- (iv) In the fourth case, the position of the robot and the goal are in different nodes (i.e., rooms) and the goal is not within the current local boundaries (see Figure 4.18). The global path is used to determine where the goal gets projected onto the potential field's border (see Section 9.3).

The four scenarios encapsulate all potential situations where the local and global path planners may interact. The role of the global path planner is to provide the necessary information to the local path planner in order to project the goal (i.e., global) onto the local path planner's border. The local path planner's borders are repositioned when the robot approaches the current local bounds, so that the robot is in the centre of the new bounds. Their extent is limited because the computation of the harmonic function is exponentially proportional to the number of grid elements and the size of a grid element can only be a certain size which is related to the context of the environment (see Section 7).



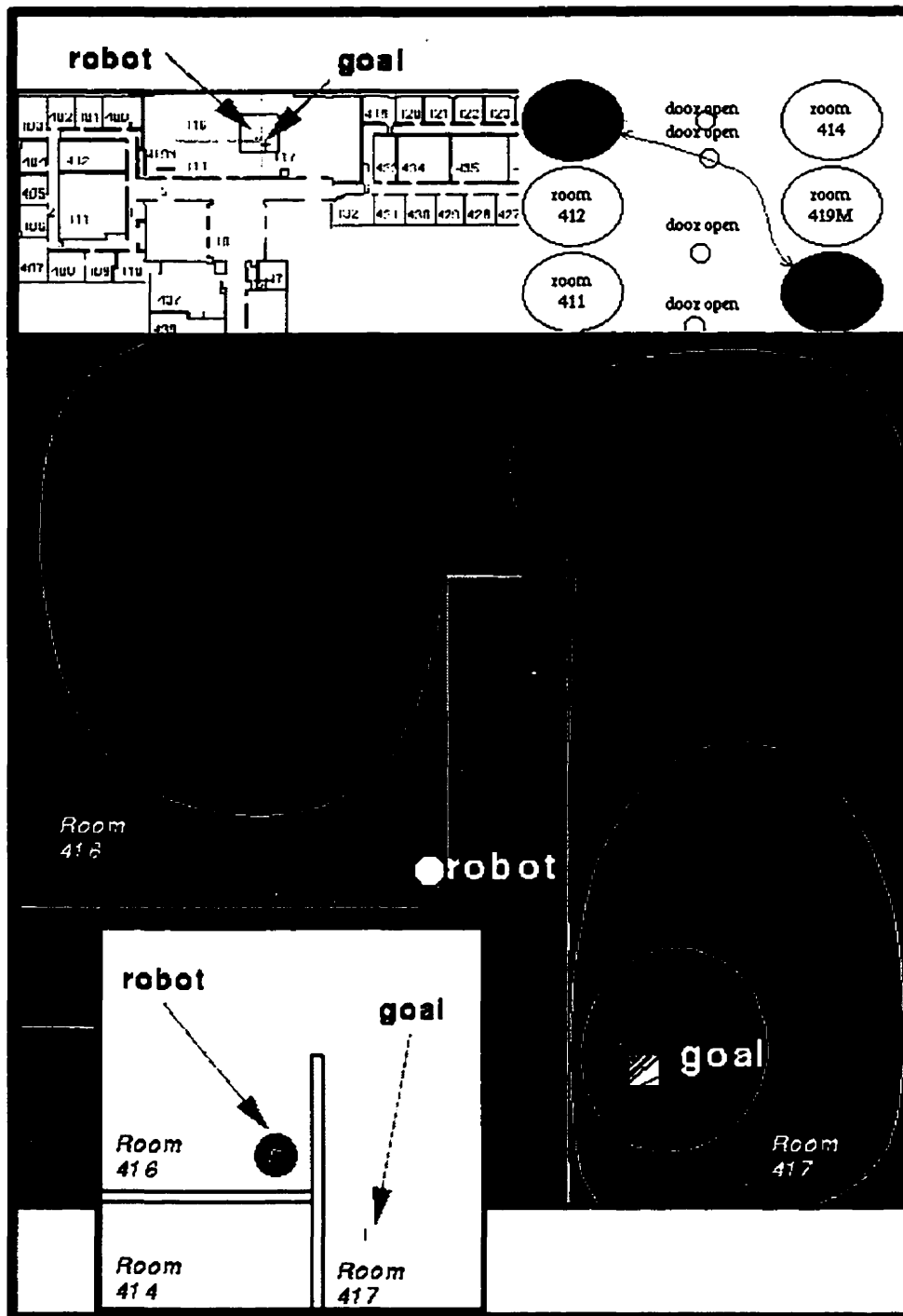
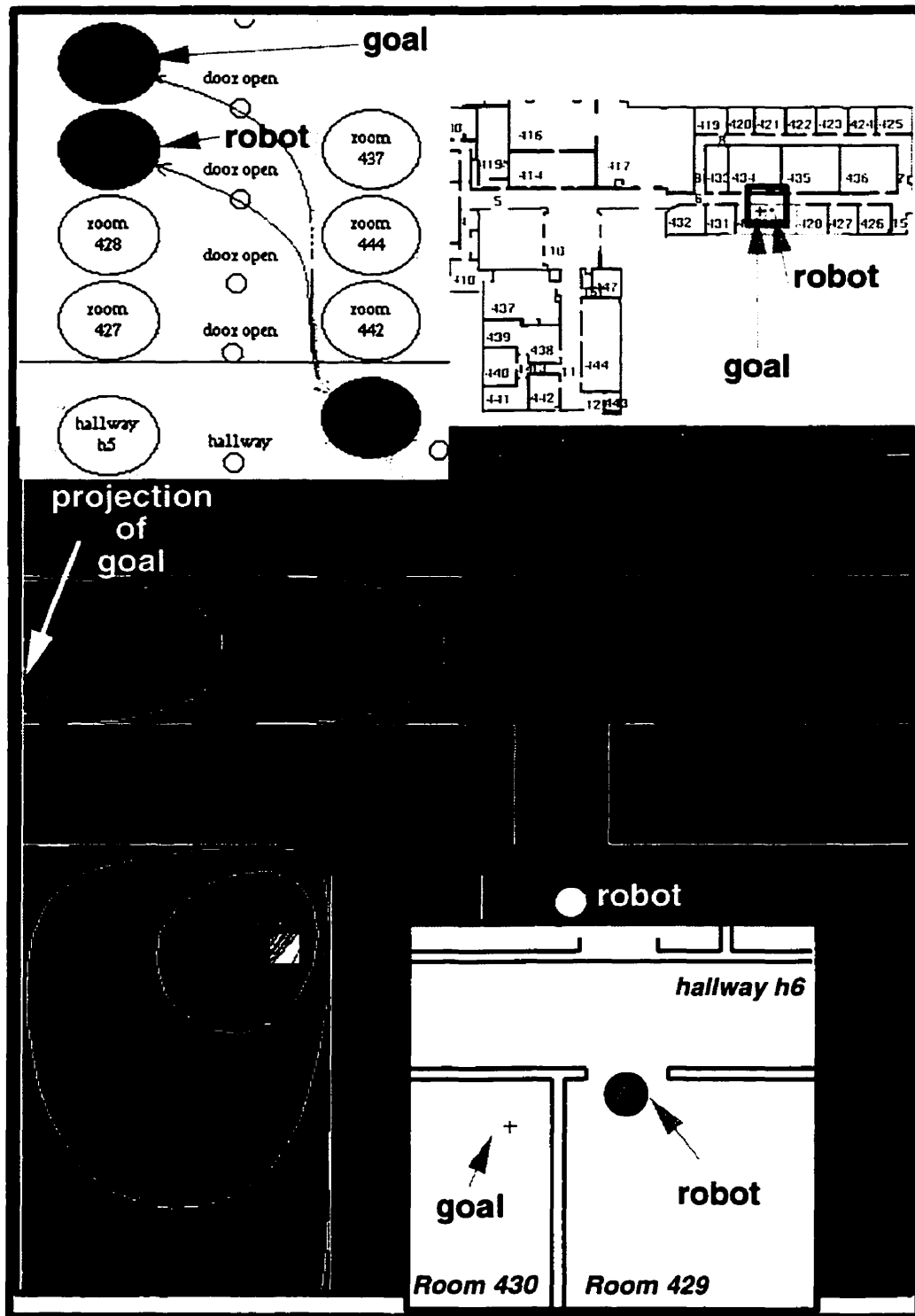


FIGURE 4.15. Local and Global Path Planner Interaction: First Typical Case. The position of the robot and the goal are within the local extent of the potential field, and the goal is reachable (i.e., there exists a path within the local confines of the potential field from the robot to the goal position) from the robot's current position.



**FIGURE 4.16. Local and Global Path Planner Interaction: Second Typical Case.** The position of the robot and goal are within the local extent of the potential field, but the goal is unreachable (i.e., there is no path within the potential field's local confines from the robot to the goal) from the robot's current position. The access way into the room, where the goal is located, is not contained within the local extent. The global path is used to determine the projection of the goal onto the proper location on the boundary of the potential field (see Section 9.3).

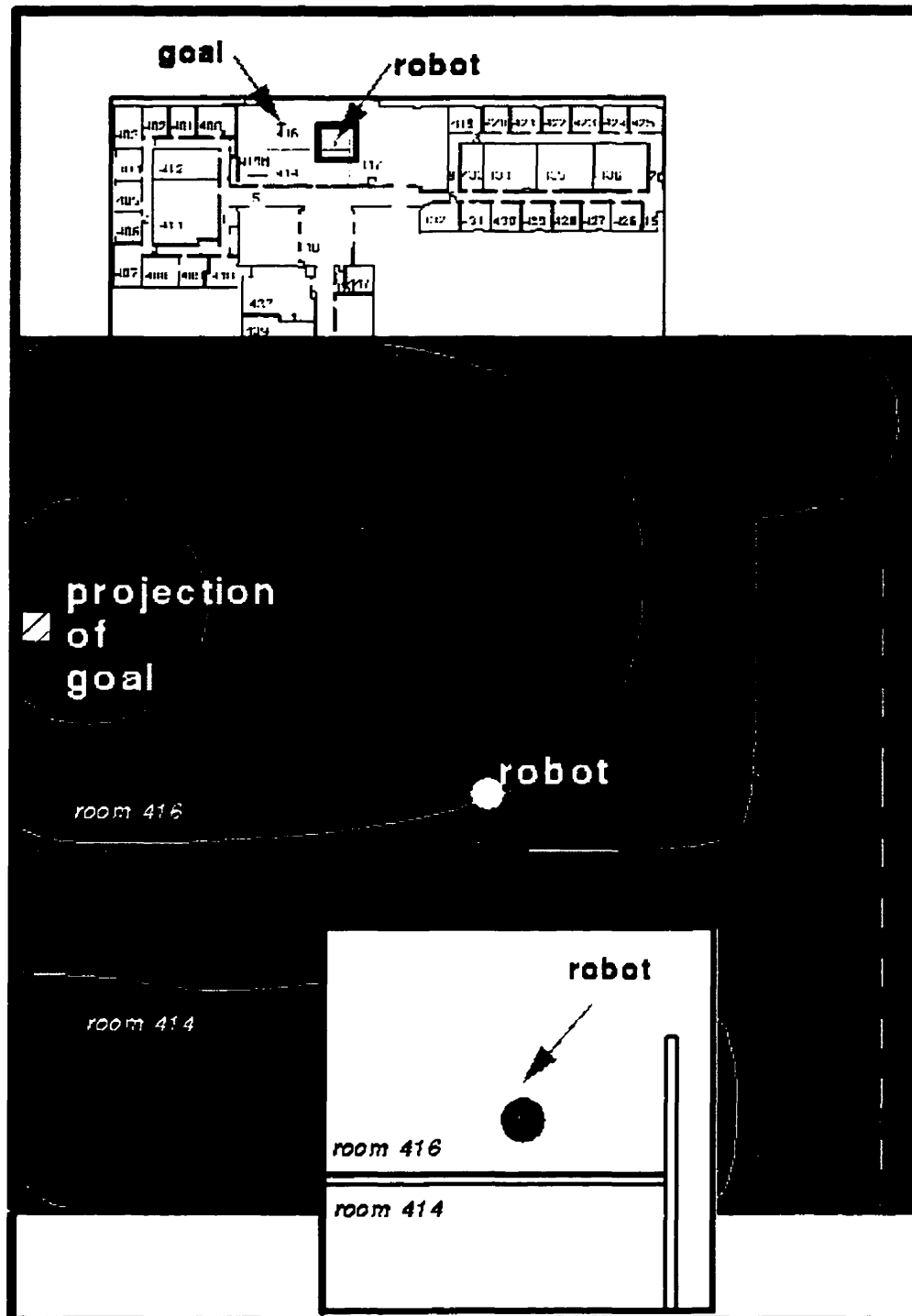


FIGURE 4.17. Local and Global Path Planner Interaction: Third Typical Case. The robot and the goal are located in the same node (i.e., room), but the goal is not within the local extent of the potential field. The intersection of a line connecting the robot and the goal with the potential field's border determines the location of the projection of the goal onto the potential field's border.

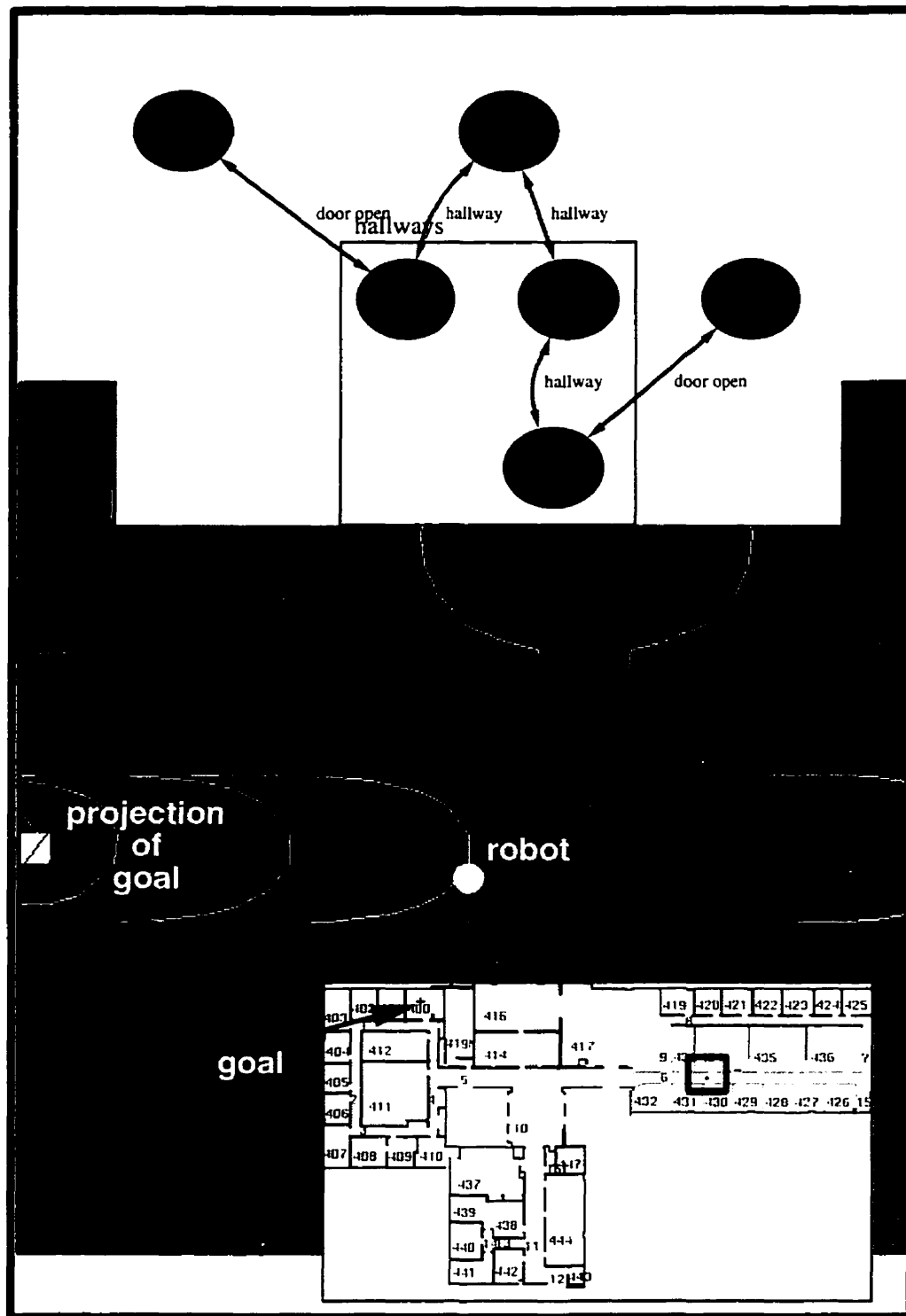


FIGURE 4.18. Local and Global Path Planner Interaction: Fourth Typical Case. The position of the robot and the goal are in different nodes (i.e., rooms). The global path is used to determine where the goal gets projected onto the potential field's border (see Section 9.3).

## 9.2. Reachability

A global goal is reachable from a particular starting position if there is at least one path to it which avoids all obstacles. Some of the obstacles are given a priori (i.e., permanent features such as walls contained in the CAD map) while others are discovered while executing the task. In normal operational mode, SPOTT is given an architectural CAD map, and a corresponding graph<sup>42</sup> as a priori information. Initially, SPOTT assumes that all the doors are open and the global path planner plans a minimum cost path from the starting node to the node containing the global goal. During the course of traversing this initial path, SPOTT senses its environment and may find doorways that are actually closed. If this doorway is part of the planned path, then at this time, the global path is replanned (see Section 8). It may be the case that the robot cannot leave its particular node or replan a path to the global goal because of the newly discovered state of the doors (i.e., closed). If this is the case, execution is ceased, and the operator is informed that the goal cannot be reached.

During execution, SPOTT is continually sensing its environment. It may be the case that the robot is not able to traverse a node because newly discovered obstacles are blocking its path within the node and the local path planner cannot plan a path around them (e.g., boxes blocking a hallway). This condition is referred to as *blocking*. Within a particular node, the robot is either traversing to an edge (i.e., doorway or virtual plane) or the global goal (i.e., if it is within the current node). One way of determining reachability of the local or global goal is to perform spatial reasoning (i.e., potentially computationally expensive) using the map (i.e., the CAD map and the newly sensed features) to determine if the goal is reachable. This task is beyond the scope of SPOTT and is left to a reasoning agent<sup>43</sup>.

A relatively inexpensive (i.e., computationally) approach has been taken to detect *blocking*, which capitalizes upon one of the features of the local path planner (i.e., potential field). If there is no path to the local goal, the potential function near the location of the position of the robot will flatten out and produce no negative gradient. If there is no negative gradient, the robot will not move. If the robot does not move after a specified period of time<sup>44</sup>, then SPOTT is notified that there is a blockage at the current location. If the current location is in the node where the robot and the global goal are both located, then the operator is notified that the robot cannot attain its goal. However, if the robot is in one of intermediate

<sup>42</sup>Consisting of nodes for rooms and hallway portions, and edges for access ways (e.g., doors).

<sup>43</sup>This is one of the future planned tasks of the logical reasoning module called COCOLOG (Caines & Wang, 1995), which has been recently interfaced with SPOTT (see Section 3 in Chapter 6).

<sup>44</sup>Four seconds was chosen for a 35 by 35 potential field. A potential field of this size should converge in about two seconds, and waiting for four seconds is ample time to determine that the goal is having no attractive influence.

nodes (i.e., part of the global path), then the global path planner is notified of the blockage so that it can try to plan another route. The potential field planner specifies this blockage as a *reachability constraint* on the replanning of the global path. The constraint is specified as  $(n_j, (x_r, y_r), flag, e_i, (x_g, y_g))$ , where  $n_j$  is the node where the blockage occurred,  $(x_r, y_r)$  is the position of the robot when it was blocked, and *flag* determines whether the next goal is an edge  $e_i$  or the global goal  $(x_g, y_g)$ .

An example of the use of *reachability constraints* by the global path planner when planning a path from node to node is shown in Figure 4.19. In this case, a blockage was previously detected in node 3 at position  $B$  when traversing node 3 on the way to edge 4. When the global path is replanned, edge 4 is ineligible as an edge in the global path depending on the location where node 3 is entered in relation to the point of blockage. If node 3 is reentered at edge 1, then edge 4 is not eligible as the next edge to traverse: however if node 3 is entered at edge 2 or 3, then it is eligible as the next edge to traverse.

A straight-line distance comparison is made between the entering point  $e_j$  and the blockage points specified by the *reachability constraints*. If the distance to a destination edge  $e_i$  from the location where the node is entered -  $e_j$  - is less than the distance from the blockage point (i.e., specified by the *reachability constraint*) to the edge  $e_i$ , then the local goal is allowed to be part of the new global path. If the distance is greater, then edge  $e_i$  is not a permissible candidate for the global path. The information conveyed by the *reachability constraint* in Figure 4.19 is that the robot cannot reach edge 4 from the point of blockage  $B$ , and therefore any locations beyond  $B$  (i.e., further away from edge 4 than  $B$  is) are also assumed to be blocked.

### 9.3. Algorithm: Projection of Global Goal onto Local Border

The following is the algorithm used for determining how a global goal is projected onto the border of the potential field:

- (i) If the goal is within the local extent of the potential field's border and in the same node that the robot is in, then do nothing.
- (ii) If the goal is in the same node that the robot is in but the goal is not within the potential field's border, then the location of the global goal is projected onto the border by joining the global goal to the robot position, and intersecting this line with the potential field's border.

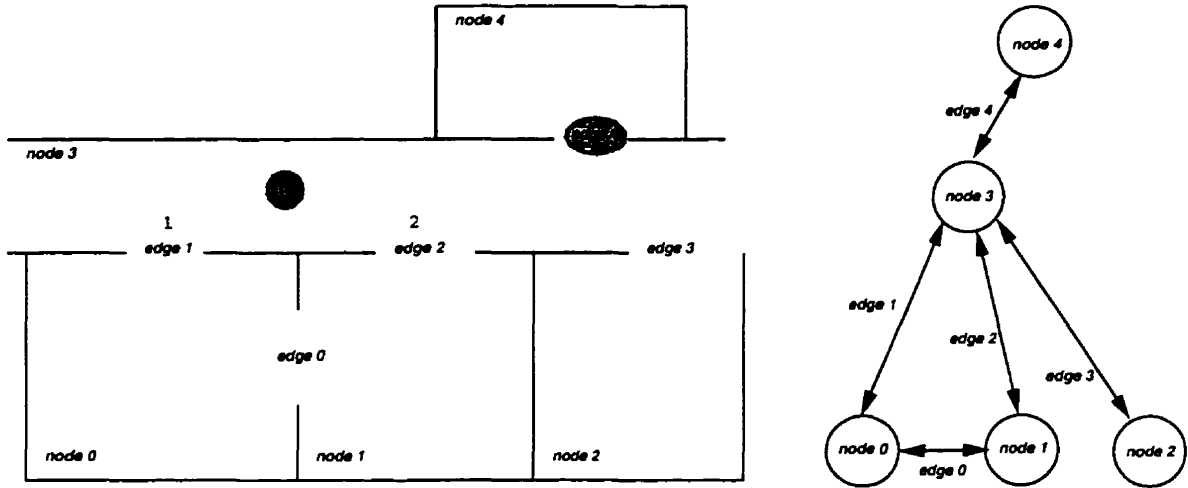


FIGURE 4.19. **Reachability and Blocking Constraints** The figure illustrates the use of a *reachability constraint* which specifies that a blockage was previously detected in node 3 at position *B* when traversing node 3 on the way to edge 4. When the global path is replanned, edge 4 is ineligible as an edge in the global path depending on the location where node 3 is entered in relation to the point of blockage. If node 3 is reentered at edge 1, then edge 4 is not eligible as the next edge to traverse: however if node 3 is entered at edge 2 or 3, then it is eligible as the next edge to traverse. The information conveyed by the illustrated *reachability constraint* is that the robot cannot reach edge 4 from the point of blockage *B*, and therefore any locations beyond *B* (i.e., further away from edge 4 than *B* is) are also assumed to be blocked.

- (iii) Otherwise, the projection is based on finding the first edge within the global path that is not traversable (i.e., *locally reachable*<sup>45</sup>) by the robot while staying within the local confines of the potential field. The projection is based on joining the centre of that edge to the centre of the node last traversed before reaching that edge.

The global path is specified as  $\{n_i, n_{i+1}, n_{i+2}, \dots, n_n\}$ , where  $n_i$  is the start node and  $n_n$  is the goal node. The edges of the global path are  $\{e_{(i,i+1)}, e_{(i+1,i+2)}, \dots, e_{(n-1,n)}\}$ , where the indices correspond to the two nodes that the edge connects. The goal is *locally reachable* if the robot can safely pass through all of the edges<sup>46</sup> that are part of the global path, while staying within the bounds of the potential field. If the edge is outside the local extent, then the edge is not passable. If the edge intersects the border of the potential field, then the part of the edge inside the local extent requires to be larger than the width of the robot for the edge to be passable. If the edge is labelled as a closed door, then it is not passable. Otherwise, if the edge is completely within the local extent of the potential field and is labelled as an open door or a virtual plane, then the edge is passable.

<sup>45</sup>An edge is *locally reachable* if the robot can fit through the portion of the access way that is in the current local extents of the potential field.

<sup>46</sup>Each edge represents a line in a 2D map, which corresponds to a door or virtual plane in 3D space.

The global path is used to determine the projection of the goal onto the proper location on the boundary of the potential field. This is accomplished by finding the first access way (i.e., edge), proceeding in sequential order in the list of edges  $\{e_{(i,i+1)}, e_{(i+1,i+2)}, \dots, e_{(n-1,n)}\}$ , that is not passable (i.e., locally)  $e_{(i+k,i+k+1)}$ . This edge corresponds to an access way between nodes  $n_{i+k}$  and  $n_{i+k+1}$ . The intersection of a line connecting the centre of the edge defined by  $e_{(i+k,i+k+1)}$  and the centre of the part of the node  $n_{i+1}$  that is within the potential field's borders (or the location of the robot if it is in that node), with the potential field's border is used to determine the projection of the goal (see Figure 4.21).

Figures 4.20 through to 4.22 show some examples of the global goal being projected onto the local potential field border.

- In Figure 4.20, the robot and the goal are located in the same node (e.g., room), but the goal is not within the local extent of the potential field. The intersection of a line joining the robot and the goal with the potential field's border is used to determine the goal projection.
- In Figure 4.21, the position of the robot and goal are within the local extent of the potential field, but the goal is locally unreachable from the robot's current position. The projection is based on the intersection of a line defined by two points with the potential field border. The first point is the centre of the part of node *h6* that is within the potential field bounds. The second point is the centre of the edge that connects nodes *h6* and *room 430*. This second point is also the centre of the first edge that is not locally passable along the way from the start to goal node.
- In Figure 4.22, the position of the robot and the goal are in different nodes (i.e., rooms) and the goal is not within the current local boundaries. The global path is used to determine where the goal gets projected onto the potential field's border so as to locally maintain the execution of the global path.

The topological path produced by the global path planner is used to project the global goal onto the potential field's border (i.e., if it is necessary). This co-operation permits the local path planner (i.e., potential field) to overcome its shortcomings in extent (i.e., the size of the potential field's extent is limited because an increase in size is directly proportional to its computation time) by providing it with information about the global goal. In a static environment, this arrangement will permit the global goal to be attained. This is because the projection algorithm is based on tracing the global path's geometric



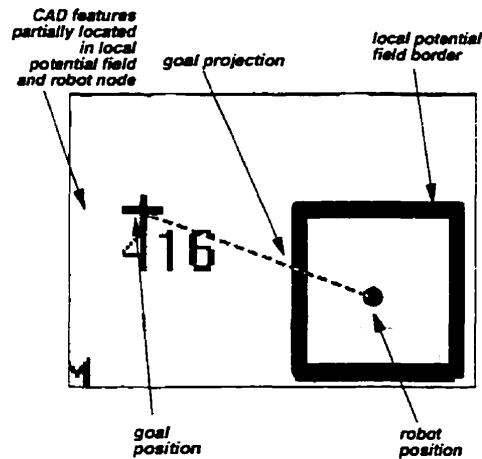


FIGURE 4.20. Goal Projection: Goal in Same Node but not in the Potential Field. The robot and the goal are located in the same node (e.g., room), but the goal is not within the local extent of the potential field. The intersection of a line connecting the robot and the goal with the potential field's border is used to determine the goal projection. The shaded walls in this diagram and subsequent ones correspond to CAD wall features that are at least partially in the potential field border or in the node in which the robot is located.

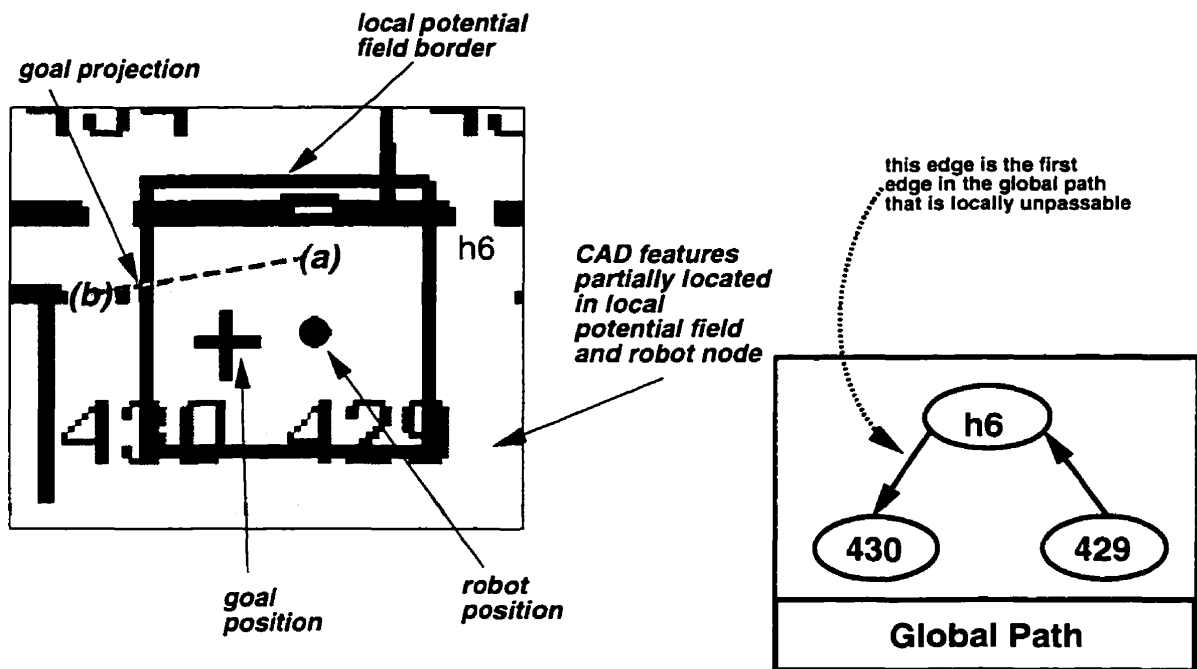


FIGURE 4.21. Goal Projection: Locally Unreachable Goal. The position of the robot and goal are within the local extent of the potential field, but the goal is locally unreachable from the robot's current position. The projection is based on the intersection of a line defined by two points with the potential field border. (1) The first point of the line is the centre of the part of node *h6* that is in the potential field bounds (i.e., (a)). (2) The second point of the line is the centre of the edge that connects nodes *h6* and room *430* (i.e., (b)). The second point is also the centre of the first edge that is not locally passable along the way from the start to the goal node.

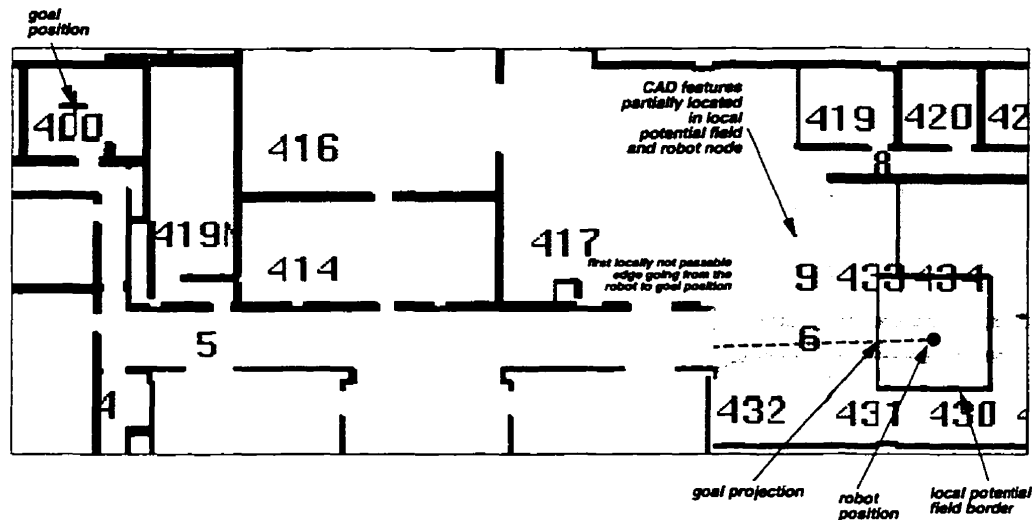


FIGURE 4.22. Goal Projection: General Case. The position of the robot and the goal are in different nodes (i.e., rooms) and the goal is not within the current local boundaries. The global path is used to determine where the goal gets projected onto the potential field's border so as to locally maintain the execution of the global path.

correspondences of the nodes and edges within the local confines of the potential field<sup>47</sup>. In a dynamic environment, obstacles can be discovered anywhere (i.e., unknown a priori) and only experimentation will determine what the limitations<sup>48</sup> of the projection algorithm are for dynamic path planning in indoor environments or other environments where global structure can be captured in an abstract graph (i.e., of nodes and edges).

<sup>47</sup>The abstract graph's nodes are rectangular, thus simplifying the geometrical projections of the global goal.

<sup>48</sup>One potential shortcoming occurs when an obstacle is discovered at the potential field boundary, just in front of the location where the goal has been projected to. This will have the effect of erasing the attractive pull of the projected goal. A heuristic method of overcoming this is to make the projected point a line (i.e., a line covering a portion of one of the sides that comprise the boundary of the potential field). Even with this method, one can imagine sensing objects that will block the attractive pull of the projected goal. The proper way to overcome this limitation is to have a spatial reasoner reproject the goal when such a situation occurs.

## 10. Future Research Issues

It is theoretically possible to extend the formalism presented here for 2D path planning (i.e., local path planning) to address non-holonomic<sup>49</sup> control, moving obstacles, and 3D path planning. These cases are of interest because they address more general cases of the presented method: (1) Most vehicles<sup>50</sup> (e.g., automobile) are non-holonomic; (2) Moving obstacles (e.g., people, other robots) are common in most office and warehouse environments; and (3) 3D path planning is necessary for planning motions when looking for (i.e., finding) objects in 3D space and to guide a manipulator if the mobile robot is equipped with one. One way to address extension into these three areas is to include another dimension into the potential field grid space to represent: (1) constraints for non-holonomic control (Connolly & Grupen, 1994); (2) time in order to capture obstacle dynamics for control with moving obstacles; or (3) the vertical dimension for 3D path planning. The addition of another dimension, while theoretically possible, is not practical in most cases<sup>51</sup>. The computational time for a harmonic function on a discretized grid will increase in direct proportion to the number of grid points. The extension of the presented techniques for path planning to these new areas is a research challenge that requires a practical solution given current technology.

---

<sup>49</sup>A *non-holonomic* mechanical system is one whose motion in configuration space is locally constrained (Connolly & Grupen, 1994). For example, a wheeled vehicle can move to any position and orientation with a sequence of maneuvering moves but its motion at any instant is constrained (Hwang & Ahuja, 1992). This results in a constraint on the curvature of paths for a wheeled vehicle. Traditional path planning techniques treat mechanical systems as being holonomic (i.e., not constrained locally).

<sup>50</sup>The Nomadics 200 mobile robot is non-holonomic, but it is treated as a holonomic system.

<sup>51</sup>It is practical only when the discretized grid is coarse in 2D and will be coarse in the third dimension also (i.e., there are not that many grid points).

## CHAPTER 5

---

### A Task Command Language Lexicon

Interaction between robots and humans should be at a level which is accessible and natural for human operators. There has been very little research done pertaining to human-machine natural language interaction and communication in the field of autonomous mobile robot navigation (Lueth *et al.*, 1994). Natural language permits information to be conveyed in varying degrees of abstraction subject to the application and contextual setting. A major function of language is to enable humans to experience the world by proxy, “*because the world can be envisaged how it is on the basis of a verbal description*” (Johnson-Laird, 1989). There are two aspects to a natural language interface: (1) speech recognition and its interpretation into a vocabulary lexicon; and (2) a vocabulary lexicon and its interpretation into control and representational constructs. The research area of speech recognition is beyond the scope of this thesis and is not addressed, and only the interpretation into control constructs for SPOTT is discussed.

SPOTT’s language lexicon is a minimal spanning subset for human 2D navigational tasks (Landau & Jackendoff, 1993; Miller & Johnson-Laird, 1976). The task command lexicon consists of a verb, destination, direction and a speed. The destination is a location in the environment defined by a geometric model positioned at a particular spatial location in a globally-referenced Cartesian coordinate space. The basic form of the task command lexicon is a variation of a *command* or *request* (Hodges & Whitten, 1986) sentence and its syntax is:

$$\begin{array}{rcl}
 & & \text{VERB} \\
 & + & \text{(from)SOURCE} \\
 (5.5.1) \quad \text{COMMAND} = & + & \text{(to)DESTINATION} \\
 & + & \text{DIRECTION} \\
 & + & \text{SPEED}
 \end{array}$$

If it is assumed that the robot's starting location is its current location, the reference to the *SOURCE* can be removed which leaves the following as the basic form:

$$\begin{array}{rcl}
 & & \text{VERB} \\
 (5.5.2) \quad \text{COMMAND} = & + & \text{(to)DESTINATION} \\
 & + & \text{DIRECTION} \\
 & + & \text{SPEED}
 \end{array}$$

User-specified commands are formulated using the lexicon template given by Equation 5.5.2. Internal communications and representational constructs can also be expressed using parts of the lexicon set.

The tasks are based on the verbs “*GO*” and “*FIND*”, and a mode of operation called *intelligent tele-operation*. A task command formulated with the verb *GO* assumes that the spatial location of the goal is known, whereas a task command formulated with the verb *FIND* assumes that a description of the object is known but its spatial location is not. *Intelligent tele-operation* is concerned with navigating the robot in a specific direction with no predefined target - following simple directional commands such as *forward*, *left*, to name a few - until an obstacle is encountered.

The description of the spatial location for the “*GO*” task is a 2D geometric primitive situated at a particular location in the 2D navigational coordinate space. The modelling of an object is subject to the representations used for 2D path planning, which are 2D geometric models<sup>1</sup> (i.e., points, lines, ellipses, rectangles). These 2D geometric models are used as goals for SPOTT's local path planning module (see Chapter 4). Typically, a point (i.e., geometric object) situated in the 2D coordinate space is used to specify a goal.

The kind of objects SPOTT can “*FIND*” is limited by its a priori knowledge and perceptual capabilities. Currently, QUADRIS is used to recognize walls, doors, and chairs (Bui, in preparation). A range profile is used to recognize these objects. Current research (Bui, in preparation) is also looking into the recognition of geons (i.e., general 3D primitives, see Figure 5.2). using a collection of range profiles.

Another element of the task command is a minimal spanning subset of prepositions (Landau & Jackendoff, 1993) that are used to spatially modify goal descriptions<sup>2</sup> (e.g., near, behind), and to specify trajectory commands (e.g., left, right, north). The spatial relationships used are sparse, primarily including qualitative distinctions of distance and direction. The quantification of the spatial and trajectory prepositions depends on two

<sup>1</sup>These 2D geometric models may be the result of a projection onto the 2D navigational plane of a corresponding 3D model representing an object (e.g., door, chair, wall) in the environment.

<sup>2</sup>Goals are specified by a spatial region in a global coordinate reference frame.

norms: the definitions for the spatial prepositions *near* and *far* in the current environment and task context<sup>3</sup>. In language design, the descriptors (e.g., spatial prepositions) filter out metric information (i.e., not explicitly encoded), and similarly, such descriptions may be instrumental for providing the structure for a level in a cognitive map<sup>4</sup>. The task command is interpreted and quantified by SPOTT, and subsequently used as input to a TR+ program (see Chapter 3) and the local path planner (see Chapter 4).

---

<sup>3</sup>It may be a very difficult task to quantify the definitions for *near* and *far* based on the current environment and task context. For example, *far* can either mean that the object (e.g., pen) is not reachable or that the object is somewhere in the horizon (e.g., mountain).

<sup>4</sup>SPOTT's map database does not currently use spatial prepositions to encode topological relationships, although, such descriptions might be useful for a symbolic reasoner.

## 1. Verbs

Verbs in the English language can be classified into categories of *motion*, *possession*, *vision* and *communication* (Miller & Johnson-Laird, 1976). *Motion* and *vision* are the two categories that are of interest for specifying a navigational task command.

### 1.1. Motion Verbs

Verbs of motion describe how people and things change their places and orientations in space (Miller & Johnson-Laird, 1976). They describe how an object changes from a place  $p_1$  at time  $t_1$  to another place  $p_2$  at a later time  $t_{1+i}$ . The word *travel* appears to capture this idea of a location change at least as well as any other single verb in the English language (Miller & Johnson-Laird, 1976).

*Deictic* words are the type where it is necessary to know the conditions under which a word occurs in order to interpret it (Miller & Johnson-Laird, 1976). The deictic verbs of motion in English are a small set (e.g., *bring*, *come*, *go*, *send*, *take*), but are the most frequently used in common speech. When the verb *go* contrasts with *stay* (i.e., the nondeictic sense of *go*), it is closely similar in meaning to *travel*. *Go* is chosen as the verb to mean *travel* because the syntax of the task command (i.e., command, request) implies the nondeictic use of the word *go*, and *go* is more commonly used in everyday speech. This is also the verb used by Jackendoff (1990) to capture the concept of travel.

### 1.2. Vision Verbs

The conceptual core of vision rests on the verbs *look* and *see* (Miller & Johnson-Laird, 1976). *Look*<sup>5</sup> is behavioral (i.e., involves the act of moving the eyes or a part of the body) while *see* is experiential (i.e., perceptual). The central paradox of perception is that “*no claim about veridical visual perception can be visually verified*” (Miller & Johnson-Laird, 1976). The individual is the only person to know what it is that they perceive: other people cannot verify this component of a veridical claim.

Verbs that are based on *look* are behavioral in nature. These verbs involve the concept of a visual process in which a sequence of perceptual acts is under the control of an observer. *Looking over* a perceptual object is such a process. The sort of things that can be looked over include areas, surfaces, and objects. These verbs involve a role of intention and can be classified as follows:

---

<sup>5</sup> *Look* is defined as “*to set one’s eyes upon something or in some direction in order to see*”. *See* is defined as “*to perceive with the eyes; to perceive things mentally, discern, understand; to construct a mental image of*”. Both definitions are taken from the *Webster’s Encyclopedic Unabridged Dictionary*.

- Visual monitoring may be impossible unless the perceptual system is intentionally guided to act in order to continue looking at a perceptual object (i.e., *look at*).
- An observation may be deliberately prolonged in order to see what happens to the perceptual object (i.e., *watch for*).
- A series of observations may be initiated in order to come to be able to see a particular entity (i.e., *look for*, *search for*, or *find*).

The verb *find* involves both looking and seeing. In this case, looking is moving the eyes (i.e., controlling the pan-tilt head of a camera) and body (i.e., robot) in a strategic search sequence, and seeing is the perceptual process of identifying the sensed input as one of the recognizable entities sought out by the agent (i.e., robot). *Watching for something* and *searching for something* are both perceptual activities and motivated by a higher-order intention to learn something. When one is watching, the target is usually an event or an action, but when one is searching, it is usually an object. An object is found if it becomes *visible*, which in essence is also an event. In this sense, *find*<sup>6</sup> is used to search for an object or event.

*Find* is the perfect companion to the word *go* in the verb slot of the task command. It specifies the case where a description of an object is given, but its spatial location is unknown.

---

<sup>6</sup>The perceptual sense of *find* is also directly related to the control instruction *find*.



## 2. Spatial Locations and Object Descriptions

Using *go* and *find* as the verbs of a task command lexicon implies that the destination for the robot is either a known spatial location (i.e., *go*) or an unknown place specified (i.e., *find*) by only an object's description (e.g., a ball).

A known destination is a spatial region that is positioned in a predefined Cartesian coordinate space. Since SPOTT's task set is based on 2D navigational tasks, a set of simple 2D models (i.e., point, line, ellipse, rectangle) are used to model the spatial area defining the destination (see Figure 5.1). In the real world, objects are 3D in shape and a proper

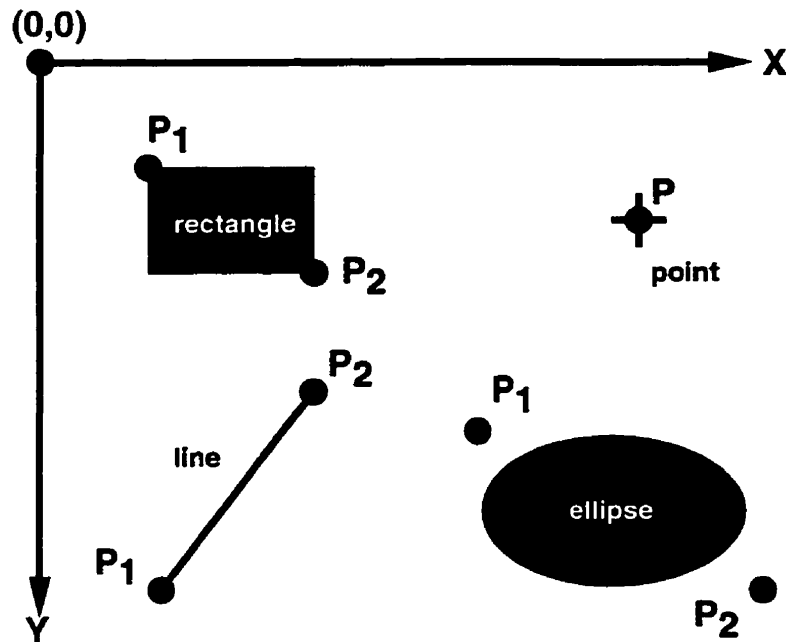


FIGURE 5.1. 2D Models for Representing the Spatial Area Defining the Destination. The four different types of models used include a point, line, ellipse and rectangle. A point is specified by a  $P = (x, y)$  coordinate pair. The rest of the models are specified by a set of two points  $P_1 = (x_1, y_1)$ ,  $P_2 = (x_2, y_2)$ , defining the top left and bottom right points. An ellipse is specified by the two points defining a bounding rectangle.

description for perceptual identification would be a 3D model<sup>7</sup>.

Parametric geons (Wu & Levine, 1993) are chosen as a qualitative description of object components for future use when SPOTT's perceptual capabilities have advanced. Parametric geons have been shown to be applicable for qualitative object recognition (Wu, 1996). There are seven of them (i.e., ellipsoid, cuboid, tapered cylinder, tapered cuboid, curved cylinder, and curved cuboid) and they are defined by parametrized equations where the size

<sup>7</sup>The 2D models are just projections of the 3D models onto the 2D navigational plane.

and degree of tapering and bending is controlled (see Figure 5.2. Equations 5.5.3 through to 5.5.9, and (Wu & Levine, 1993)).

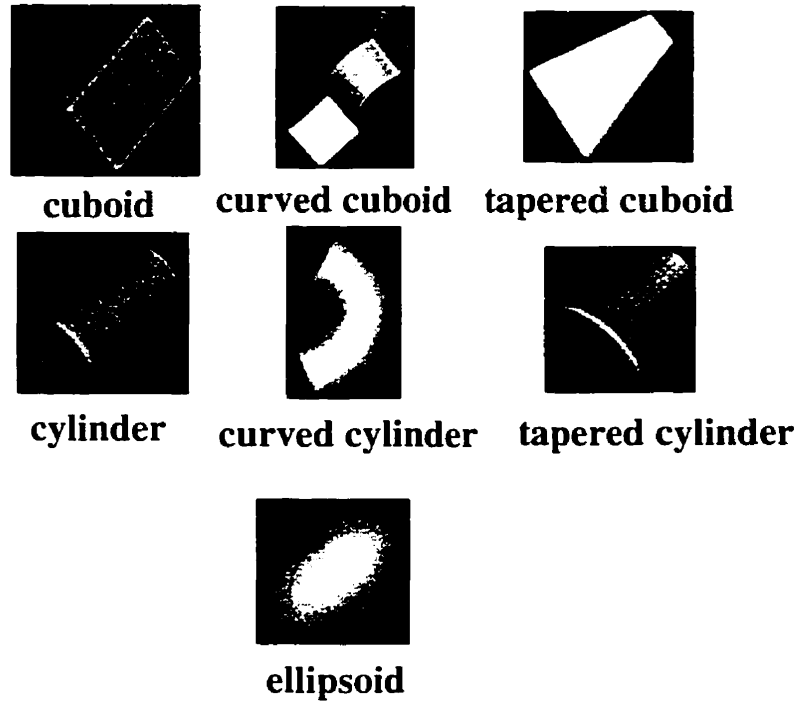


FIGURE 5.2. Geons. Seven qualitative shape types defined by parametrized equations that control the size and degree of tapering and bending have been proposed (Wu & Levine, 1993) as a qualitative description of object components. These shape types are used to describe 3D objects found in the world. For example, a cuboid (i.e., thin plate) would describe a wall. Other objects may require the combination of two or more shape types to describe their shape (e.g., a chair would be described by a thin plate for the backing and seat and four long cylinders for the legs).

The equation of an *ellipsoid* (Wu & Levine, 1993) is:

$$(5.5.3) \quad \left(\frac{x}{a_1}\right)^2 + \left(\frac{y}{a_2}\right)^2 + \left(\frac{z}{a_3}\right)^2 = 1.$$

The equation of a *cylinder* (Wu & Levine, 1993) is:

$$(5.5.4) \quad \left(\left(\frac{x}{a_1}\right)^2 + \left(\frac{y}{a_2}\right)^2\right)^{10} + \left(\frac{z}{a_3}\right)^{20} = 1.$$

The equation of a *cuboid* (Wu & Levine, 1993) is:

$$(5.5.5) \quad \left(\frac{x}{a_1}\right)^{20} + \left(\frac{y}{a_2}\right)^{20} + \left(\frac{z}{a_3}\right)^{20} = 1.$$

The equation of a *tapered cylinder* (Wu & Levine, 1993) is:

$$(5.5.6) \quad \left( \left( \frac{X}{a_1(\frac{K_x}{a_3}Z + 1)} \right)^2 + \left( \frac{Y}{a_2(\frac{K_y}{a_3}Z + 1)} \right)^2 \right)^{10} + \left( \frac{Z}{a_3} \right)^{20} = 1$$

where  $K_x$  and  $K_y$  are tapering parameters. The equation of a tapered cuboid (Wu & Levine, 1993) is:

$$(5.5.7) \quad \left( \frac{X}{a_1(\frac{K_x}{a_3}Z + 1)} \right)^{20} + \left( \frac{Y}{a_2(\frac{K_y}{a_3}Z + 1)} \right)^{20} + \left( \frac{Z}{a_3} \right)^{20} = 1$$

The equation of a *curved cylinder* (Wu & Levine, 1993) is:

$$(5.5.8) \quad \left( \left( \frac{\kappa^{-1} - \sqrt{Z^2 + (\kappa^{-1} - X)^2}}{a_1} \right)^2 + \left( \frac{Y}{a_2} \right)^2 \right)^{10} + \left( \frac{\kappa^{-1} \arctan \frac{Z}{\kappa^{-1} - X}}{a_3} \right)^{20} = 1$$

where the curvature value  $\kappa$  is used to describe a simple bending operation around a centre of a circle centered on the x-axis in the x-z plane (Wu & Levine, 1993). The equation of a *curved cuboid* (Wu & Levine, 1993) is as follows:

$$(5.5.9) \quad \left( \frac{\kappa^{-1} - \sqrt{Z^2 + (\kappa^{-1} - X)^2}}{a_1} \right)^{20} + \left( \frac{Y}{a_2} \right)^{20} + \left( \frac{\kappa^{-1} \arctan \frac{Z}{\kappa^{-1} - X}}{a_3} \right)^{20} = 1$$

The types of 3D models currently in use are constrained by the perceptual capabilities of the robot. SPOTT's recognition capabilities are limited to walls, doors, and chairs (Bui, in preparation). These types of objects may be described by a collection of adjoining cuboids. This kind of object description (i.e., as a collection of geon parts) is useful for the perceptual process of recognition. In order to describe a place, usually the location is idealized by a single part (3D or 2D) (Herskovits, 1985).

### 3. Prepositions

The preposition is a key element in the linguistic expression of place and path, and there are few of them in comparison to the number of names for objects (Landau & Jackendoff, 1993). The totality of prepositional meanings is extremely limited. This fixes the number of prepositions in the English language (see Table 5.1) which makes them ideal for use in a robot task command language<sup>8</sup>. There are two different types of prepositions that are of interest for a robot task command lexicon: (1) one type describes a spatial relationship between objects; and (2) the other describes a trajectory. A preposition in its spatial role is only an ideal. The actual meaning is a deviation from the ideal. It is determined by the context of a specific application. A level of *geometric conceptualization* mediates between *the world as it is* and language (Herskovits, 1985).

A locative expression is any spatial expression involving a preposition, its object and whatever the prepositional phrase modifies (noun, clause, etc.):

$$(5.5.10) \quad NP_1(\textit{preposition})NP_2$$

where  $NP$  is a *noun phrase*. If a noun phrase ( $NP_i$ ) refers to an object ( $O_i$ ), then the locative expression can be rewritten as follows:

$$(5.5.11) \quad IM(G_1(O_1), G_2(O_2))$$

where  $G_i$  is the geometric description applied to the object ( $O_i$ ), and  $IM$  is the ideal meaning of the preposition. The ideal meaning of a preposition is such that (1) it is manifested in all uses of the preposition, although shifted or distorted in various ways, and (2) it does not apply to the referents of the noun-phrase, but to geometric descriptions associated with these referents (Herskovits, 1985). If  $[T(IM)]$  is the transformed ideal meaning, then the geometric scene representation can be stated as follows:

$$(5.5.12) \quad [T(IM)](G_1(O_1), G_2(O_2))$$

The different types of categories that typify a geometric description function are as follows (Herskovits, 1985):

- (i) Functions that map a geometric construct onto *one of its parts*. Examples of parts include a 3D part, edge, or the oriented base of the total outer surface.
- (ii) Functions that map a geometric construct onto *some idealization of the object*. The idealization can be an approximation to a point, line, surface, or strip.

---

<sup>8</sup>Spatial prepositions may also be used for qualitatively describing the spatial relationships amongst parametric geons that are used to model the parts of a single object (Landau & Jackendoff, 1993).

<i>Spatial Prepositions</i>					
about	above	across	after	against	along
alongside	amid(st)	among(st)	around	at	atop
behind	below	beneath	beside	between	betwixt
beyond	by	down	from	in	inside
into	near	nearby	off	on	onto
opposite	out	outside	over	past	through
throughout	to	toward	under	underneath	up
upon	via	with	within	without	
<i>Compounds</i>					
far from			in back of		
in between			in front of		
in line with			on top of		
to the left of			to the right of		
to the side of					
<i>Intransitive Prepositions</i>					
afterward(s)	apart		away		
back	backward		downstairs		
downward	east		forward		
here	inward		left		
N-ward (e.g., homeward)	north		outward		
right	sideways		south		
there	together		upstairs		
upward	west				
<i>Nonspatial Prepositions</i>					
ago	as	because of	before	despite	during
for	like	of	since	until	

TABLE 5.1. **The Minimal Spanning Set of English Prepositions.** The possible meanings of all prepositions is extremely limited (Landau & Jackendoff, 1993). The above lists the set of all prepositions that minimally span all the prepositional meanings possible in the English language.

- (iii) Functions that map a geometric construct onto some associated *good form*. A good form is obtained by filling out some irregularities or implementing Gestalt principles of closure or good form.
- (iv) Functions that map a geometric construct onto *some associated volume that the construct partially bounds*. Adjacent volumes include the interior, the volume or area associated with a vertex, and lamina associated with a surface.
- (v) Functions that map a geometric construct onto *an axis, or a frame of reference*.
- (vi) Functions that map a geometric construct onto *a projection*. The projections can either be a projection on a plane at infinity or a projection onto the ground.

The meaning is transformed due to various contextual factors<sup>9</sup> bearing on the choice and interpretation of a location expression (Herskovits, 1985).  $T(IM)$  and  $G$  functions are frequently fuzzy: however, quantification of a spatial prepositional expression into a fuzzy definition is difficult because context is the key and its quantification is difficult (Herskovits, 1988).

Understanding the representations of space requires invoking mental elements corresponding to places and paths, where places are generally understood as regions often occupied by landmarks or reference objects. The spatial preposition is an operator that takes as its arguments, both the figure object and the reference object, and the result of this operation defines a region in which the figure object is located:

$$(5.5.13) \quad F_{sp}(O_f, O_r) = R$$

where  $F_{sp}$  is the spatial preposition defining a function operating on the figure  $O_f$  and reference  $O_r$  object models (i.e.,  $O_f$  is  $O_1$  and  $O_r$  is  $O_2$  in Equations 5.5.11 and 5.5.12) in order to obtain a region of interest  $R$ .

The use of spatial prepositions may impose certain constraints on the figure or reference objects, or the resulting region. The restrictions placed on the form of the reference object or figure object by spatial prepositions are not very severe, and only refer to the gross geometry at the coarsest level of representation of the object (Landau & Jackendoff, 1993). The object's axial structure plays a crucial role. In a spatial expression defining a location, there are no prepositions which cause the figure or reference object to be analyzed in terms of a particular geon<sup>10</sup> (Wu & Levine, 1993; Biederman, 1987). In general, there are no

<sup>9</sup> *Saliency* explains the direction of metonymic shifts. For example, one can use a noun which basically denotes a whole object to refer to a part of it which is typically salient. *Relevance* has to do with communicative goals, with what the speaker wishes to express or imply in the present context. *Tolerance* is the deviation from the truth of the ideal meaning, usually talked about in measurable terms such as angle or distance. *Typicality* is the implication of other meanings associated with the use of the current preposition (e.g. the use of *behind* implies nearness).

<sup>10</sup> A qualitative description of an object component (i.e., part) (see Section 2).

prepositions that insist on analysis of the figure or reference object into its constituent parts. A reference object can be schematized as a point, a container or a surface, as a unit with axial structure, or as a single versus aggregate entity. A figure object can be schematized as most as a single lump or blob (no geometric structure whatsoever), a unit with axial structure that is along, at most, one of its dimensions, or a single versus distributed entity.

### 3.1. Quantifying Prepositional Expressions

The spatial relations encode several degrees of distance and several kinds of direction. Many complexities arise from assigning different frames of reference. Some spatial expressions involving axes do not leave this choice of reference system open. The choice of axes can either be defined by the reference object<sup>11</sup>, the speaker (i.e., operator), or the actor (i.e., robot).

The quantification of spatial prepositions depends on the purpose of the use. One possible method is to quantify a spatial expression into regions with a degree of membership, which is done by *fuzzy sets* (Zadeh, 1974). This representation is useful as an information source for search strategies that determine the likelihood of locating an object in a particular region, which is encoded by the spatial prepositional expression. At this time, SPOTT is interested in encoding spatial prepositions in a task command lexicon in order to define regions in space to navigate to. Therefore, the quantification requires a precise location or region in a global coordinate space.

Recall that SPOTT's path planner's navigational space is two-dimensional (see Chapter 4) and the geometric descriptors of a goal are a point, line, ellipse (e.g. circle) and a rectangle (e.g. square) (see Section 2). The regions describing the result of a spatial prepositional expression are described with respect to the boundaries of these geometric objects. The four levels of distance that are described by English spatial prepositions (Landau & Jackendoff, 1993) are:

- (i) location in the region interior to the reference object (e.g. *in, inside*),
- (ii) location in the region exterior to the reference object but in contact with it (e.g. *on, against*),
- (iii) location in the region proximate to the reference object (e.g. *near*), and
- (iv) location distant from the reference object (e.g. *far, beyond*).

Besides the boundary defined by the reference object, the quantification of a spatial expression only depends on a norm for describing what is meant by *near* and *far*. Although

<sup>11</sup>A reference object can have a usual orientation associated with it (e.g., the front of the chair is the direction where one can sit from).

humans are able to represent distance at finer levels for tasks that require finer control, it appears that spatial prepositions do not encode this precision.

Vandeloise (1991), in describing the French equivalents of the English spatial prepositions *near* and *far*, claimed that they are often described in terms of the following factors:

- (i) The access of the target<sup>12</sup>,
- (ii) The dimension of the landmark and to a lesser extent, the size of the target.
- (iii) The size of the speaker, and
- (iv) The speed of the target<sup>13</sup>.

The norm that defines *near* and *far* needs to be defined with respect to some context. Denofsky (1976) claimed that the norm for *far* can be approximately equal to four times the norm for *near*; however, this is clearly not applicable in all situations<sup>14</sup>. For mobile robot navigation, some categories (i.e., or context) that can help quantify the definition are manipulator access, perception access, and the bounds of the environment:

- Manipulator access can define the norm for *near* as being the maximum reach of the manipulator arm from the body surface of the robot.
- Perception access can define the norm for *far* as being equal to the furthest distance that the robot can perceive objects.
- A bounded environment's (e.g., a room) maximum dimensions can define a norm for *far*.

Even though there are only two norms to define (i.e., *near* and *far*), it is not clear that such simple definitions for their quantification are adequate across different contextual settings. Currently in SPOTT, the operator specifies the norms for *near* and *far* that are subsequently used in the quantification of a spatial expression in the task command.

Only 2D spatial expressions are used in SPOTT's task command and the vertical orientation prepositions are not currently included in the vocabulary, because all tasks are on a planar surface. SPOTT's task command lexicon categorizes the prepositions it uses into three groups:

- (i) Spatial prepositions that depend on a orientation based on the reference object (i.e., directional prepositions) as well as the norms for *near* and *far*,

<sup>12</sup>Access can either be physical (i.e., reachability for defining *near*) or perceptual (i.e., defining *far* based on the furthest recognizable object).

<sup>13</sup>If the target is moving towards the landmark, the norm may increase with the speed of the target. If the target is moving away from the landmark, the extent of the norm will diminish as the speed decreases. The speed of the landmark may also be a factor: however, it is uncommon to find a landmark that is mobile with respect to the target. The speed of the speaker, and the size and the speed of the addressee may also be factors.

<sup>14</sup>For example, in an outdoor setting, the picnic tables may be *near* and the distant mountains may be considered *far*.



- (ii) Spatial prepositions that depend only on the distal norms for *near* and *far* with no need for any orientation frame (i.e., distal prepositions), and
- (iii) Prepositions that define a trajectory (i.e., trajectory prepositions).

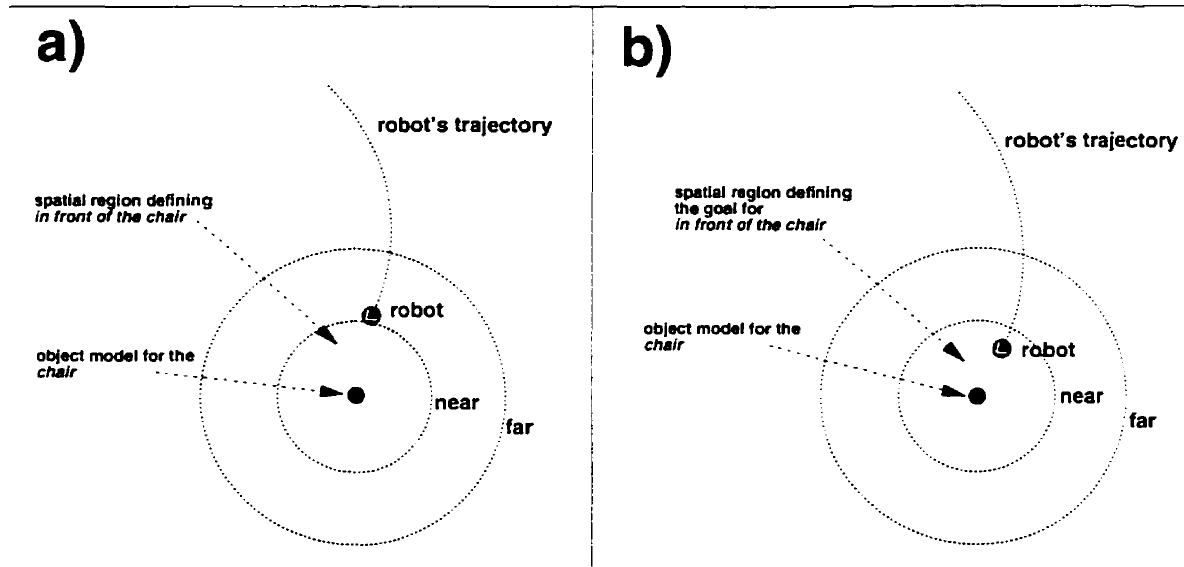
Both the directional and distal prepositions are used as part of the *destination* member of the task command. The trajectory prepositions are used to define the *direction* member of the task command (see Equation 5.5.2 and Figure 5.8).

### 3.1.1. Directional and Distal Prepositions

The result of quantifying the spatial expression (i.e., in the *destination* part of the task command) is to define a goal region for the path planner (i.e., potential field, see Chapter 4). In the potential field (i.e., path planner), the entire goal region is saturated to a low potential value and is not used as part of the free space, which is where the robot navigates. The robot navigates by performing steepest gradient descent on the harmonic function which is computed in the free space. The robot will stop when it reaches the outskirts of the goal region. Ideally, the robot should navigate towards the centre of the region defined by a spatial expression. This is why the region which is used to define the goal region is only half the size of the region defined by the spatial expression (see Figure 5.3).

Figure 5.4 illustrates graphically how the set of directional prepositions are quantified to act as a goal (i.e., for path planning) for different 2D object models (i.e., point, line, ellipse, rectangle). The choice of axes can either be defined by the reference object or the speaker (i.e., the robot) or by the reference object (e.g., the front of the chair is the direction where one can sit from). The prepositional region is based on the chosen orientation axis, the orientation denoted by the spatial prepositions, as well as the norms for *near* and *far*. The quantification of the spatial expression into a goal region is accomplished by taking the half of the region defined by the spatial expression which is closest to the object model (see Figure 5.3).

Figure 5.5 illustrates graphically how the set of directional prepositions are quantified to act as a goal (i.e., for path planning) for different 2D object models (i.e., point, line, ellipse, rectangle). The set of distal values are only four in number: interior, contact, proximal (i.e., near) and distal (i.e., far). Distal prepositions differentiate themselves from directional prepositions by not being oriented.



**FIGURE 5.3. Spatial Expression Regions Defined as Goals for Path Planning.** The result of quantifying the spatial expression (i.e., in the *destination* part of the task command) is to define a goal region for the path planner (see Chapter 4). This figure illustrates the region defined by the spatial expression “*in front of the chair*”, and the final stopping position of the robot when this spatial expression is used as the goal. The goal can be quantified in two different ways: in (a), the region defined by the spatial expression is used as the goal for the path planner; and in (b), only half the region is used as the goal. The rationale behind using only half the region is that the robot traverses to the outskirts of the goal region, and in (b) the robot will end up in the middle of the region defined by the spatial expression. This contrasts with (a) where the robot will end up at the outskirts of the region defined by the spatial expression.

2D GEOMETRIC MODEL PREPOSITION LEGEND N = near F = far backward right left forward propositional region potential field object	+	—	●	■
IN FRONT OF ALONG *				
IN BACK OF ACROSS *				
BEHIND				
BEYOND				
TO THE RIGHT OF				
TO THE LEFT OF				
TO THE SIDE OF ALONGSIDE BESIDE BY				

FIGURE 5.4. Directional Preposition Definitions. The goal of quantifying the spatial expression is to define a goal region for the path planner (see Chapter 4). The first row is the set of 2D object models used. The second row shows how the *near* and *far* definitions relate to the different 2D object models. The subsequent rows show the region used as a goal model (i.e., *potential field object*) when the object model in question is acted upon by the spatial preposition found in the first column. The legend for the graphics and symbols used is shown in the box situated in the first column and second row. There are two regions highlighted throughout the table: the *propositional region* and the *potential field object region*. The region used to describe the result of the spatial expression is the one highlighted as the *potential field object* together with the region highlighted by the *propositional region*. The *potential field object* region is the one used as the goal model for the spatial expression.

2D GEOMETRIC MODEL PREPOSITION LEGEND N = near F = far backward right ← left forward propositional region potential field object	+	—	●	■
NEAR AT AGAINST AROUND				
FAR				
INSIDE IN				
OUT OUTSIDE				

FIGURE 5.5. Distance Preposition Definitions. The goal of quantifying the spatial expression is to define a goal region for the path planner (see Chapter 4). The first row is the set of 2D object models used. The second row shows how the *near* and *far* definitions relate to the different 2D object models. The subsequent rows show the region used as a goal model (i.e., *potential field object*) when the object model in question is acted upon by the spatial preposition found in the first column. The legend for the graphics and symbols used is shown in the box situated in the first column and second row. There are two regions highlighted throughout the table: the *propositional region* and the *potential field object region*. The region used to describe the result of the spatial expression is the one highlighted as the *potential field object* together with the region highlighted by the *propositional region*. The *potential field object* region is the one used as the goal model for the spatial expression. The goal region formed with the "inside" spatial preposition acting upon a point or line geometric model is the actual 2D object model. In the case of an ellipse or rectangle, the goal region is a smaller ellipse or rectangle centered in the 2D object model. For the *far* preposition, the goal region is defined as everything but the 2D object model. Prepositions such as *in* and *out* are only relevant when the reference object can be entered by the robot (e.g., a rectangle representing a room).

### 3.1.2. *Trajectory Prepositions*

The role of the trajectory prepositions (i.e., *direction* category in the task command) is to bias the steepest gradient descent operation that is performed on the potential function by the local path planner (see Section 5 in Chapter 4). Rather than using steepest gradient descent, the trajectory is a descending gradient that is biased to be as close as possible to the direction specified by the trajectory preposition (see Figure 5.6). There are three different types of trajectory prepositions:

- For the first type, the trajectory is determined by following the outer perimeter of a reference object, keeping a distance equal to the norm for the definition of *near* away from the reference object (e.g., *along*, *around*).
- For the second type, the trajectory preposition is defined by a vector connecting the robot's current position (i.e., at the time the command is given) to the centroid of the reference object (e.g., *towards*, *away from*).
- For the third type, the quantification of the trajectory preposition is defined by a coordinate axis centered at the robot (e.g., *backward*, *forward*, *left*, *right*) or a global coordinate axis which defines the compass heading (e.g., *north*, *south*, *east*, *west*).

Figure 5.7 illustrates the direction biases that the set of trajectory prepositions define.

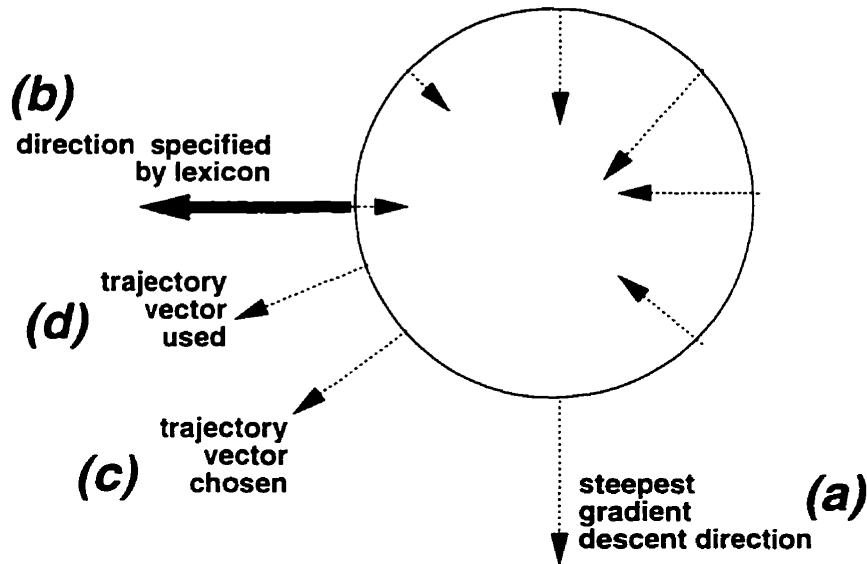


FIGURE 5.6. Trajectory Preposition Bias on Steepest Gradient Descent. The role of the trajectory prepositions (i.e., *direction* category in the task command) is to bias the steepest gradient descent operation that is performed on the potential function by the local path planner (see Section 5 in Chapter 4). Recall that the steepest descent gradient at a particular location is computed using a 3 by 3 operator. The steepest descending gradient (a) is chosen out of the nine choices (i.e., the dotted vectors). The gradient used is based on finding a descending gradient which is the closest one to the orientation specified by the trajectory preposition (b). Once this has been chosen (c), an interpolation procedure is initiated, similar to the one described in Figure 4.9, in order to find a descending gradient which is the closest in orientation to the trajectory preposition (d).

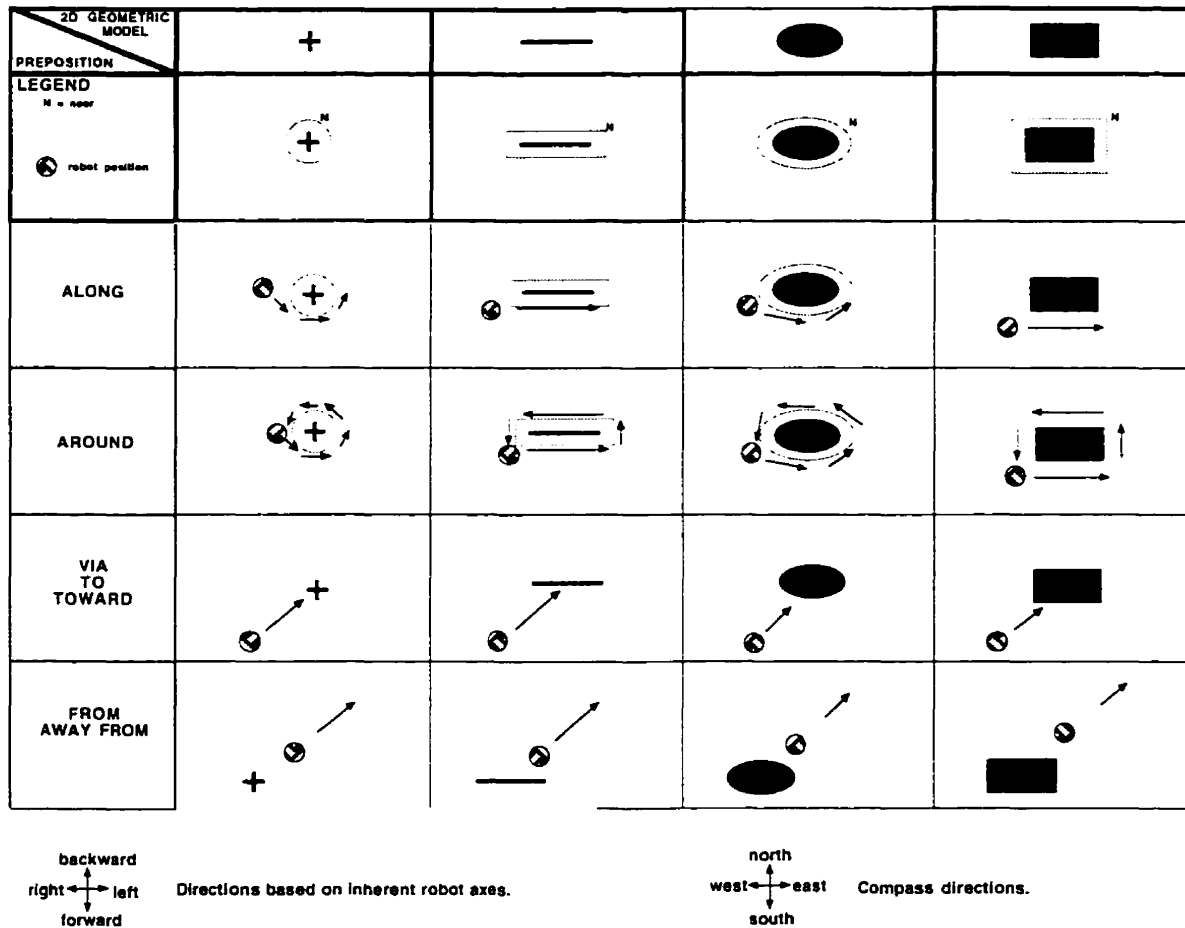


FIGURE 5.7. Trajectory Preposition Definitions. The role of the trajectory prepositions is to bias the steepest gradient descent operation (See Figure 5.6) on the potential function performed by the local path planner (see Section 5 in Chapter 4). The first row is the set of 2D object models used. The second row shows how the definition for *near* relates to the different 2D object models. The icon which indicates the robot and its position is shown in the legend box, which is in the first column and second row. The third through to the sixth rows show in terms of arrows how the trajectory of the robot is defined by the trajectory prepositions. In the third and fourth rows, the trajectory is determined by following the outer perimeter of a reference object, keeping a distance equal to the norm for the definition of *near* away from the reference object (e.g., along, around). In the fifth and sixth rows, the trajectory preposition is defined by a vector connecting the robot's current position (i.e., at the time the command is given) to the centroid of the reference object (e.g., towards, away from). The quantification of another set of trajectory prepositions (in the seventh row) is determined by a coordinate axis centered at the robot (e.g., backward, forward, left, right) or a global coordinate axis defining a compass heading (e.g., north, south, east, west).

#### 4. The Task Command

Combining all lexical elements results in the task command as shown in Figure 5.8. The basic syntax is given by a verb, destination, direction and speed choice. The targets chosen are based on what SPOTT can perceptually recognize (e.g., door, wall), what SPOTT knows about (e.g., rooms, hallways in the CAD map), 2D models, and 3D models<sup>15</sup>. Only 2D spatial prepositions are chosen because SPOTT navigates on a 2D plane. The role of specifying the direction is to bias the path planning. The lexical set of speed variables is small but a more discriminating set of speeds can be easily accommodated (e.g., very fast, very slow).

---

<sup>15</sup>3D models are planned for future use in perceptual tasks such as object recognition.



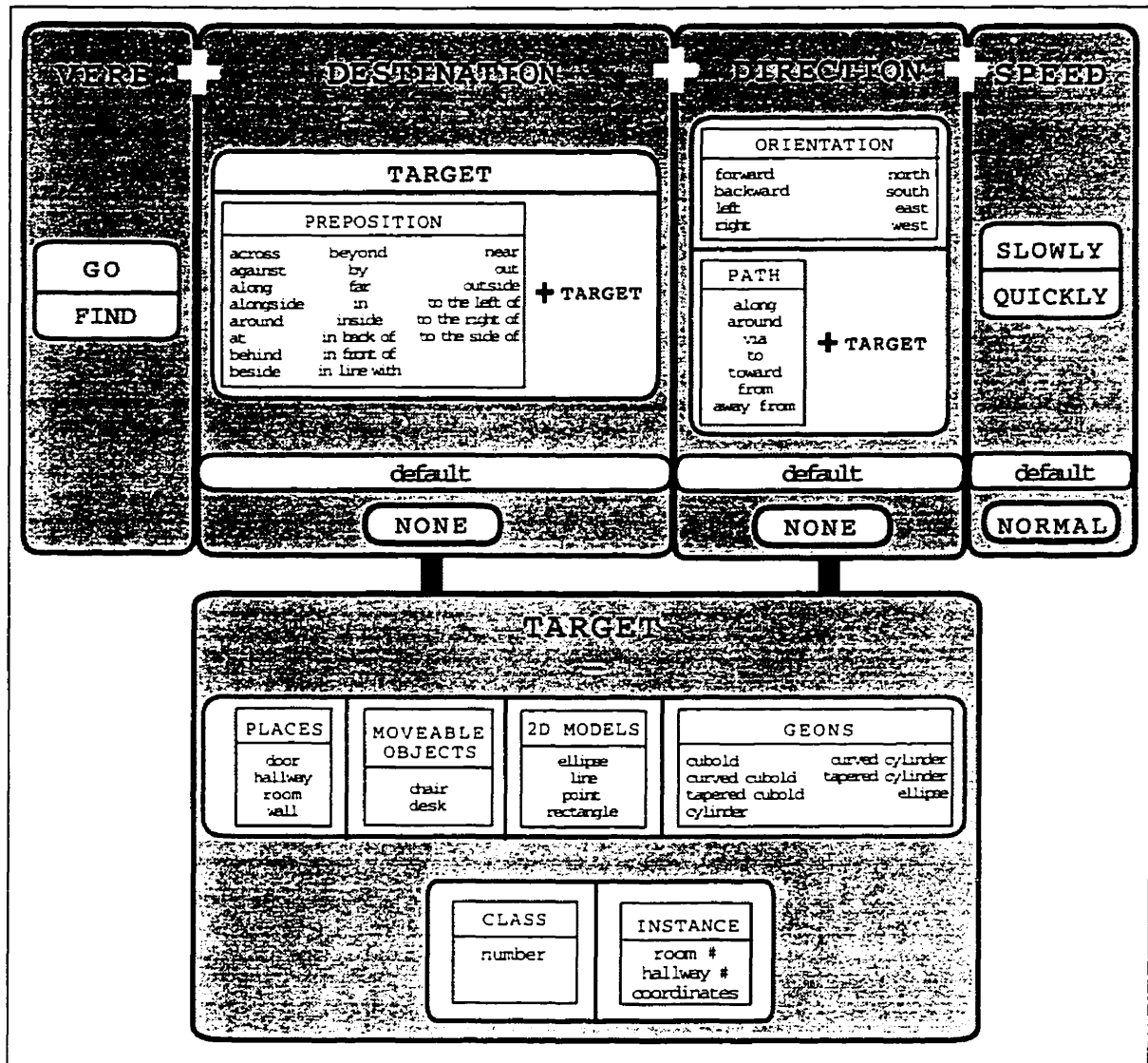


FIGURE 5.8. The Task Command Lexicon. The basic syntax is given by a verb, destination, direction and speed choice. The destination is a located region that can be modified in a spatial expression. The set of spatial preposition modifiers chosen are 2D because SPOTT is currently only able to perform 2D navigation. A set of trajectory prepositions (i.e., direction) are used to bias the trajectory defined by the local path planner.

## CHAPTER 6

---

### Implementation

SPOTT is implemented in parallel, thus permitting real-time<sup>1</sup> asynchronous functionality. The autonomous robot communicates via a radio link to SPOTT, whose processing is distributed across a collection of networked SUN and SGI workstations. The software tool PVM<sup>2</sup> (Parallel Virtual Machine) (Geist *et al.*, 1994) is used to distribute the control, planning, and graphical user interface across a collection of existing processors. PVM is a message passing library which allows the harnessing of a collection of heterogeneous processors into a single transparent, cohesive and unified framework for the development of parallel programs. The PVM system transparently handles resource allocation, message routing, data conversion for incompatible architectures, and other tasks that are necessary for operation in a heterogeneous network environment. The portability and heterogeneous property of PVM makes it possible to transfer this architecture on-board the robot in the future. The computation for both the TR+ program control and local planning is also distributed to provide real-time response. Typically, ten to fifteen processors are used in an experiment.

A distributed implementation permits SPOTT to control the robot in real-time. A parallel implementation of SPOTT is required because the computation of the potential field for the local path planner can be very computationally expensive (i.e., solving Laplace's equation iteratively with irregular boundary conditions), and some of the conditions of a TR+ program can have extensive processing of the sensor data. PVM simplifies the parallelization process by providing a high level library of function calls. The actual system calls and UNIX socket connections are transparent to SPOTT's code and are masked by a limited set of PVM library calls. Transparency between different levels of implementation are essential for understanding, maintaining and debugging complex systems such as SPOTT.

---

<sup>1</sup>Reaction time to environmental changes is faster than the rate of change in the environment.

<sup>2</sup>This is an ongoing project carried out by a consortium headed by the Oak Ridge National Laboratory.

When designing a complex system such as SPOTT, it is also important to consider visualization for system monitoring and debugging, and in general, the user (i.e., operator) interface. There are three modes of interacting with SPOTT: (1) before execution time: the operator initializes the system and specifies a task command; (2) during execution: the graphical visualization of the current actions and state of the robot within its environment continually update the visualization tools; and (3) post-mortem analysis: the operator or programmer evaluates the executed task (i.e., what went right and what went wrong). The different types of SPOTT users include:

- (i) *naive users* who are unfamiliar with the concepts that make up SPOTT.
- (ii) *partially knowledgeable users* who are familiar with the concepts that make up SPOTT (i.e., they may be able to interpret why something went wrong).
- (iii) *fully knowledgeable users* who will want to specify different initialization parameters (i.e., other than the standard) and new TR+ control programs, and
- (iv) *SPOTT system developers* who interpret the execution and enhance SPOTT by correcting newly found software bugs, improve the efficiency of execution (i.e., speed and memory management), and enhance the user interface.

SPOTT's graphical user interface tries to address all the needs for the different types of users. SPOTT is also flexible and expandable in that new TR+ control programs can be added, as well as new sensors and actuators (i.e., a robotic arm<sup>3</sup>).

PVM is the glue that has been used to implement the SPOTT system. A message-passing paradigm (i.e., PVM) has been chosen for distributing the processing because it is relatively inexpensive (i.e., existing processors are used), and the implementation is portable (i.e., operates on a collection of heterogeneous machines which can be interchanged).

---

<sup>3</sup>A manipulator arm may require its own path planner. This would require some modification to SPOTT's architecture because its current path planner is for navigational commands, and the manipulator control would require its own path planner.

## 1. Parallelism with PVM

PVM (Parallel Virtual Machine) (Geist *et al.*, 1993; Geist *et al.*, 1994; Beguelin *et al.*, 1993) is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource in a network environment. With respect to a user, a subset of the underlying network's processors are utilized so as to give the impression that the capabilities of a parallel processor are available within the confines of a single workstation. The PVM system transparently handles resource allocation, message routing, data conversion for incompatible architectures, and other tasks that are necessary for operation in a heterogeneous network environment. PVM offers excellent price-performance characteristics compared to massively parallel processors (Sunderam *et al.*, 1993). PVM was one of the first message-passing software systems<sup>4</sup> to be developed for parallelizing applications across networks. It was selected as the tool to make SPOTT concurrent because of its support (i.e., many users, news groups) and apparent robustness<sup>5</sup>, as well as ongoing research and development. The support software for PVM executes on each machine in a user-configurable pool, and presents a unified, general, and powerful computational environment for concurrent applications. PVM provides a set of C or FORTRAN user calls to the PVM library routines for functions such as process initiation, message transmission and reception, and synchronization. PVM supplies the functions to automatically start up tasks on the virtual machine and permits the tasks to communicate and synchronize with each other. SPOTT is executed on a network consisting of SUN and SGI workstations, and a PC (i.e., personal computer) 586 machine. A radio link communicates with a processor (i.e., 486) on board the robot. The implementation could be ported to be solely on board the robot if a multi-processor network were made available on the robot<sup>6</sup>.

The PVM system is composed of two parts:

- The first part is a daemon (called *pvm3*) which resides on all the computers making up the virtual machine. When a user wishes to run a PVM application, *pvm3*

---

<sup>4</sup>Other tools include Linda (Carriero & Gelernter, 1989), Isis (Birman & Marzullo, 1989), Express (Flower *et al.*, 1991), and MPI (Gropp *et al.*, 1994).

<sup>5</sup>There have been virtually no problems traced to PVM during the course of developing and testing SPOTT.

<sup>6</sup>This would be at a cost to battery life. Currently, the Nomad 200 can run for approximately four hours on a complete charge of its battery set (i.e., five batteries). For indoor environments, the processing could be done off board the robot, provided that communication is maintained between the robot and the computer network responsible for processing the control decisions. However, for remotely located environments (e.g., space applications such as the navigating on the moon or Mars) where radio transmission to the off board processing units could be slow, it would be necessary to have the computation performed on board. Experiments with the robot roaming the hallways at CIM showed that indoor transmission may also be troublesome. Transmission was lost (i.e., severely degraded) when the robot disappeared around corners (i.e., out of the line of sight of the receiver).

is executed at a *UNIX* prompt on one of the computers which in turn starts up *pvm* on each of the computers making up the user-defined virtual machine. A user configures a virtual machine (see Figure 6.1) by specifying, in a text file, the hosts which are to be part of the virtual machine. After the daemons are started, the PVM application can be started from a *UNIX* prompt on any of the machines that make up the virtual machine. Multiple users can configure overlapping virtual machines, and each user can execute several PVM applications simultaneously.

- The second part of the system is a library of PVM interface routines, used for developing application software. The PVM library is composed of user-callable routines for message passing, spawning processes, coordinating tasks, and modifying virtual machine configurations. Application programs are linked to this PVM library.

The PVM library provides routines for packing and sending messages between processes. The communication routines include an asynchronous broadcast (i.e., *send*) to a single task, and a synchronous broadcast (i.e., *multicast*) to a list of tasks. PVM transmits messages over the underlying network using the fastest mechanism available (i.e. either UDP, TCP, or other high-speed interconnects, if available). The PVM communication model also provides for asynchronous blocking *send*, asynchronous blocking *receive*, and non-blocking *receive* functions. A blocking *send* returns control as soon as the *send* buffer is free for reuse, regardless of the state of the receiver. A non-blocking *receive* returns immediately with either the data or a flag indicating if the data has arrived, while a blocking *receive* returns only when the data is in the *receive* buffer. The PVM model guarantees that message order is preserved<sup>7</sup> between any pair of communicating processes. The application developer can specify constraints on certain processes, such as constraining the computer type (e.g., SUN, SGI) or even specifying the exact machine to run the process. The default is a transparent mode, where PVM automatically chooses an appropriate machine based on the resource allocations<sup>8</sup> on the virtual machine.

The great advantage of using PVM over direct communication routines (i.e., sockets, pipes), is that application development can be done at a much higher level. The PVM software package handles the allocation of resources, and error handling, and masks the low level communication routines with the PVM library functions.

---

<sup>7</sup>Messages are processed in the order of arrival. This does not necessarily correspond to the order of time of departure from the source processes.

<sup>8</sup>The current version of PVM (i.e., 3.3.10) does not perform *load balancing*. The selection of the machine where a process is executed is based on distributing the processes evenly across the entire collection of machines that make up the parallel virtual machine.

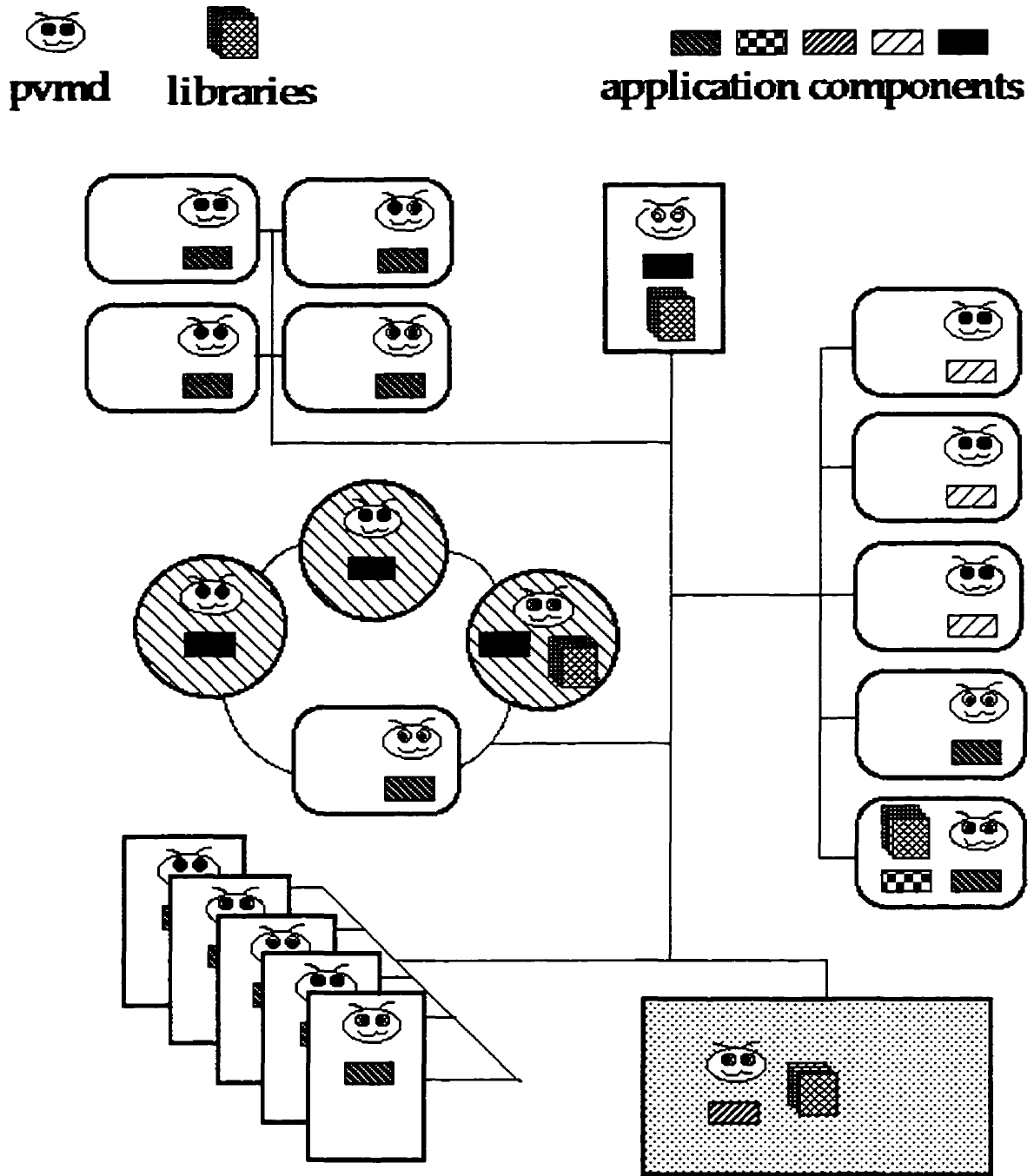


FIGURE 6.1. A Hypothetical "Parallel Virtual Machine". Each unique box shape represents an architecture. A daemon (i.e., pvmd) is running on every node. There is one set of libraries installed for each group of same-architecture nodes. There are one or more application components running on each node. The nodes are connected by different types of networks. All daemons and applications shown are for one person's run. If another person were running on the same nodes, the diagram would show another set of daemons and application programs running.

Some of the contentious issues in a message-passing parallel system are that the usable capacities of each machine on the network vary from moment to moment according to the load imposed upon them by multiple users. Another consideration is system failure: if an application executes for long periods of time, failures during execution are more likely. For example, one of the machines may crash due to a variety of reasons (e.g., another user's application, memory problems, hardware problems). If such were the case, it would be ideal to have the processes that are executing on this machine to be restarted on another machine. A system called *Dome* (Arabe *et al.*, 1995) has been recently introduced to address the problems of load balancing in a heterogeneous multi user environment, and fault tolerance. *Dome* is built upon PVM and currently is only set up for SPMD<sup>9</sup> type processing.

---

<sup>9</sup>Single Processor, Multiple Data. This is the type of computation method used by the local path planner (i.e., potential field). The first release of the software for *Dome* only became available on May 24, 1996. The potential field computation has been found to be quite robust without any current load balancing and error checking. *Dome* is not applicable for the TR+ interpreter since this is a MPMD (i.e., Multi-Processor, Multi-Data) computation style.

## 2. The SPOTT System

SPOTT is implemented as a distributed control system. The major components (i.e., processes) of SPOTT and the lines of communication between them are shown in Figure 6.2. The user (i.e., operator) interacts with the SPOTT system via the graphical user interface (GUI), and in turn the GUI interfaces to each one of SPOTT's major components: the *TR+ interpreter*, the *local path planner* (i.e., potential field), the *global path planner*, and the robot interface module (i.e., *robodaemon interface*). Besides the main GUI, there are graphical displays controlled by some of the major modules: the *TR+ interpreter* displays a graphical representation of the state of the currently executing TR+ control program; the *global graph planner* displays a graphical representation of the global path; and the *local path planner* displays the equi-potential contour lines of the potential function. In addition, the robot controlling software (i.e., *robodaemon*) has its own graphical interface for direct interaction (e.g., testing low level functionality) with the robot.

Most of SPOTT's computational resources are allocated for the processes associated with the *local path planner* and the *TR+ interpreter*. The *local path planner* computes the potential field by sending parcels of data to slave processes in a master-slave configuration (see Section 2.3). Typically, four slave processes have been used for experiments with SPOTT. This type of parallel computation style is referred to as MPSD (i.e., Multiple Processor, Single Data). In contrast, the *TR+ interpreter* uses a MPMD (i.e., Multi Processor, Multi Data) style of parallel computation. The *TR+ interpreter* spawns the computational processes that are used to compute the conditions and actions<sup>10</sup> of the currently executing TR+ control program. The condition processes acquire almost all of their sensor data (i.e., sonar, bumper, infrared, compass) from the robot via the *robodaemon* program, except for the laser rangefinder (i.e., QUADRIS) data which is acquired via a separate interface. This separate interface also permits the TR+ actions to control the pan-tilt units (PTU) of the QUADRIS system. In addition, TR+ actions send signals to the *robodaemon* process to move the robot to a particular location, and sensor data to the *local path planner* in order to update the potential field. The *GUI* process is responsible for updating the *local path planner* with the specified task command. The TR+ program is selected via the GUI and passed to the *TR+ interpreter* when the operator starts the robot via the *GUI*. Therefore, the overall controller of the SPOTT system is within the *GUI* software process.

<sup>10</sup>Currently, all actions involve only a data look up and very little computation, and are therefore executed as part of the TR+ interpreter process. In contrast, most conditions involve extensive processing of the sensor data and are implemented as separate processes.



The SPOTT system has also been interfaced to the logical control (and theorem proving) system called COCOLOG (Caines & Wang, 1995). At this time COCOLOG (see Section 3) provides, as input to SPOTT, a procedural list of high level goals (i.e., spatial locations). The relative ease in which COCOLOG was interfaced with SPOTT illustrates modularity, and the potential for expansion with the SPOTT system.

There is a graphical and computational aspect to SPOTT's implementation. The graphical interfaces are either controlled by the GUI process or one of the other central processing centres (e.g., *TR+ interpreter*) (see Figure 6.2). The other central processing centres include the *TR+ interpreter*, the *local path planner*, the *global path planner*, in addition to the low level robot and sensor interface software (i.e., *robodaemon*).

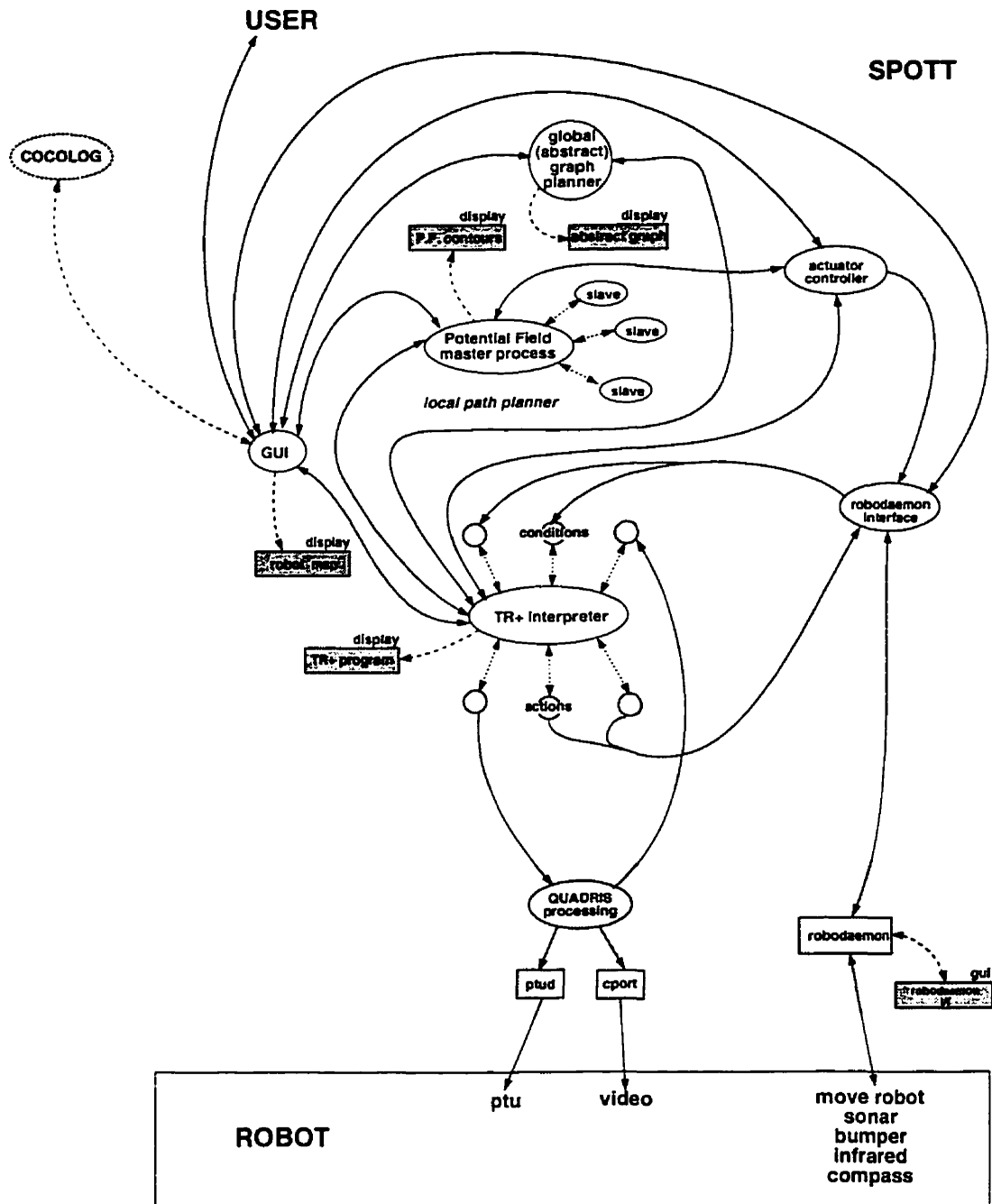


FIGURE 6.2. Implementation Modules. The major components of SPOTT, their associated processes, and the lines of communication between them are shown. Each bubble is a separate process. SPOTT's major components are: the *GUI*, the *TR+ interpreter*, the *local path planner* (i.e., potential field), the *global path planner*, and the robot interface module (i.e., *robodaemon interface*). Most of the GUI and the main control is initiated by the *GUI* process. Other components update aspects of the GUI which are pertinent to describing their computations (e.g., equi-potential contours are displayed by the local path planner). COCOLOG (Caines & Wang, 1995) is able to interface to the *GUI* by providing a procedural list of task goals (i.e., spatial locations).

## 2.1. Graphical User Interface

During execution, there is a wealth of on-line graphical information displayed, namely, the current state of the TR+ control programs, the equi-potential contours of the potential function used by the local path planner, the current path in the global (i.e., abstract) graph map, the planar map of the robot and its environment, as well as the graphical user interface for specifying the task command (and start and stop commands) (see Figure 6.3). The software package PVaniM (Topol *et al.*, 1994) is also used to provide online visualization (i.e., of processor and memory usage) of SPOTT's distributed execution with PVM. All of the graphics are displayed on two workstations<sup>11</sup>, because there is not enough display space on a single workstation. Future research will have to address the interaction with the user, in particular, the management and layout of the graphical information.

The task command is specified in a menu-based graphical interface which was developed using the software package called *Forms* (Zhau & Overmans, 1995). A planar map view of the robot's current position, executed path and its map<sup>12</sup> is also displayed and updated on a continual basis during execution. The state of the currently executing TR+ program is also displayed in its corresponding graphical notation. The local path planner is responsible for updating a display showing the equi-potential contour lines of the potential function. The contour lines are redrawn when there are modifications made to the map database: specifically, when changes are introduced into the local window that the local path planner uses. The background colour of the potential function display correlates to how well the potential function has converged to its solution. A dark hue of pink corresponds to convergence, while a light hue corresponds to the early stages of iterative computation. The colour mapping is based on taking a Laplacian operator at each discrete element location in the potential function and correlating the result with a colour look up containing increasing darker hues of pink.

The visualization software for displaying graphs was developed using the *Dotty* (Koutsosios & North, 1994) software package (see Section 2.1.1). This package is used to display the state of the currently executing TR+ program, and the current state of the path in the global graph map.

One outstanding research issue which requires further attention is how to lay out the graphical information on the computer screen and present it to the operator. Figure 6.3

<sup>11</sup>PVaniM graphics are displayed on their own display unit.

<sup>12</sup>Based on an a priori CAD map and sensed environmental information. The displayed map is either the entire map (i.e., global) known to SPOTT or a local window into the global map which corresponds to the local extent of the local path planner (i.e., potential field).

illustrates one such arrangement. However, the global path is only displayed when the operator expands the window associated with it. Additional graphics for visualization are usually displayed on another workstation (e.g., PVaniM for visualizing PVM memory and processor resource usage). The presentation of the graphics information has only been superficially addressed by this thesis and requires future examination.

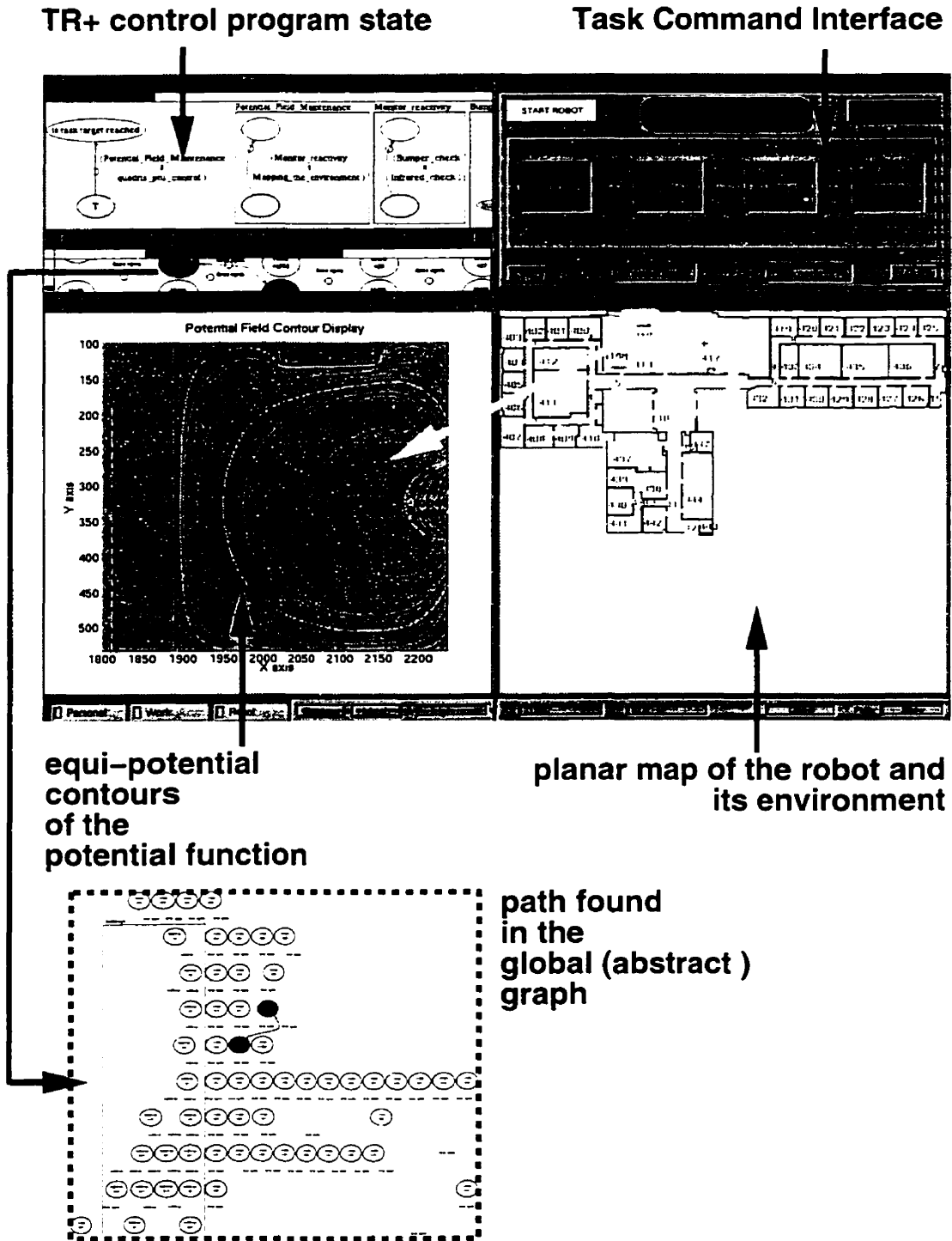


FIGURE 6.3. SPOTT's Graphical User Interface. The graphical user interface consists of five major components: (1) the *Task Command* interface from which the task command is specified; (2) the robot map which is a planar map view that is continually updated during execution; (3) a graphical visualization of the currently executing TR+ program; (4) a graphical visualization of the global graph map and the currently executing global plan (i.e., path); and (5) a equi-potential drawing of the contours in the potential function used by the local path planner (i.e., potential field). The global path (i.e., (4)) is only displayed when the user expands the window associated with it.

### 2.1.1. Drawing Graphs with Dotty

*Dotty* is a toolkit developed by AT&T Bell Laboratories which is used to create software for visually programming and monitoring graph structures. The package produces aesthetically pleasing graph layouts when provided with the graph's contents. The graphical display tools provided by AT&T<sup>13</sup> include a customizable user interface for drawing and editing graphs, a UNIX filter program for generating graph layouts based on a graphical language and graph layout algorithms, and a language for developing user interfaces and picture drawing. The major software components of the toolkit are as follows:

- *Dot* (Gansner *et al.*, 1993) is a UNIX batch filter that reads attributed graphs and writes drawings in PostScript, FrameMaker MIF, or as attributed graphs with layout coordinates specified. *Dot* uses various graph layout algorithms (Gansner *et al.*, 1993; North, 1993) to produce a graph which is easy to read. *Dot*'s graph language is very simple, and contains three kinds of items: graphs, nodes, and edges. The user is also able to specify attributes of these entities (e.g., label, colour).
- *Dotty* (Koutsofios & North, 1994) is an interactive and programmable front-end (i.e., WYSIWYG<sup>14</sup>) which runs under the UNIX X windows environment. *Dotty* is written in the *lefty* language (Koutsofios & Dobkin, 1991) (i.e., a graphical display language) and uses *Dot* as a back end to produce the graph layouts. *Dotty* includes functions to insert and delete nodes and edges, as well as functions to specify their attributes. There is also a function for computing the graph layout (i.e., performed by *dot* in the background). *Dotty* consists of a collection of non-interactive tools and filters<sup>15</sup> as opposed to a system running from a central GUI. *Dotty*'s script language is more abstract than the "C" programming language, and therefore more productive for user-interface design. Its disadvantage is that it is non-standard.
- *Lefty* (Koutsofios & Dobkin, 1991) is a general-purpose programmable editor for technical pictures with an interpretive programming language. The user interface and graph editing operations for *dotty* are written as *lefty* functions. *Lefty* is also a two-view graphics editor for pictorial (i.e., technical) pictures: (1) a view of the picture, and (2) a textual view of the program that describes the picture. Each picture is described by a program which contains functions to draw the pictures

<sup>13</sup>The graph drawing package is called *dotty*. The UNIX filter program for generating graph layouts is called *dot*. The graphical language is called *lefty*.

<sup>14</sup>What You See Is What You Get.

<sup>15</sup>*Dotty* is run through scripts, and the portability of the system permits the same scripts to be run on different platforms (e.g., SUN, SGI).

and functions to perform editing operations. User actions like mouse and keyboard events are bound to functions in a *lefty* program.

In the implementation of SPOTT, *Dotty* has been customized for the following three display tasks:

- (i) A modified version of *Dotty* is used to display the current state of the executing TR+ program. The state is continually updated every three seconds<sup>16</sup>.
- (ii) Another copy of *Dotty* has been modified to act as a user interface for creating and editing (i.e., programming) TR+ programs. This program is called "TR+edit", and was integrated with a menu-based graphical interface to make it easier for the programmer to specify the attributes of the condition and action entities which make up a TR+ program.
- (iii) *Dotty* is also used to display the abstract graph used by the global path planner. The nodes represent rooms or rectangle portions of hallways and the edges represent access ways between the nodes (i.e., doors). Highlighted nodes and edges represent the currently executing global path.

## 2.2. Using the GUI as SPOTT's Main Controller

The SPOTT system is started and controlled by its *GUI* (Graphical User Interface) process. The algorithm used by SPOTT's *GUI* is as follows:

- (i) All of the processes that are to be constantly (i.e., never stopped) running during SPOTT's execution are started and initialized: namely, the *potential field* processes - *master* and a collection of *slaves* -, the *TR+ interpreter* process, the *actuator controller* process, the *global graph planner* process, and the *robodaemon* interface process. Before SPOTT starts execution, it is assumed that the robot (i.e., Nomadics 200) and the *robodaemon* process (i.e., interface to the robot) have been started.
- (ii) A graphical events loop is initiated. Control from the user (e.g., task command specification) is event driven. Within the graphical events loop there are five main modes of operation corresponding to the execution state of the physical robot (as illustrated in Figure 6.4): (1) the robot is stopped, (2) the transition from stop to start (which involves some initialization), (3) the robot is executing (i.e., started), (4) the transition from executing mode to stopped mode (which also involves some initialization), and (5) the shutting down of the SPOTT system. The transition states

<sup>16</sup>This time interval was found to be sufficient for following the execution of a TR+ program. Continual renewal (i.e., for each cycle through the TR+ interpreter) was found to use up all of resources on the workstation displaying SPOTT's graphical components. This slowed the response time for the entire SPOTT system.

(i.e., (2) and (4)) are only visited for a single iteration through the graphical events loop. During the visitation of the transitory states, certain processes are started or stopped. Most state transitions are initiated by the operator pressing the start or stop control buttons in the *Task Command* interface portion of the GUI. There are two exceptions. The first exception is when the TR+ interpreter signals that the currently executing TR+ program has achieved its task command, which initiates the transition from the “*robot executing*” state to the “*stop initialization*” state, and subsequently to the “*stopped*” state. The second exception is when the global path planner determines that the global goal cannot be reached (i.e., it is blocked). The user may stop the entire SPOTT system in all states. During the “*robot executing*” state, the user may not specify a new task or operational parameters.

In all normal operating states<sup>17</sup> during the graphical events loop in SPOTT’s *GUI* process, the following operations<sup>18</sup> are possible based on the polling to see if a particular condition exists:

- (i) If the *actuator controller* has moved the robot, update the position of the robot.
- (ii) If the border of the local path planner (i.e., potential field) has to be moved<sup>19</sup>, then erase the current contents of the potential field and load the contents of the new local region<sup>20</sup>.
- (iii) The *potential field* master process may be notified to update its equi-potential contour display. This occurs if a new task has been specified by the operator, the map has been updated with newly sensed data, or the robot is positioned in a location such that the local potential field border has to be moved (i.e., the robot has moved near the current local bounds).
- (iv) If the robot moves to a new node in the abstract global map, then update the global path planner with this information.
- (v) If the *TR+ interpreter* has any update information for the GUI (e.g., newly sensed information, updated actuator information), then read in this information.

<sup>17</sup>With the exception of the following states: “*start SPOTT*”, “*SPOTT system is stopped*”, and “*exit SPOTT*”.

<sup>18</sup>The reason why these states are not encapsulated as a TR+ program (i.e., graph) is because a TR+ program is event driven as opposed to being strictly state driven. For the main control loop, a state driven approach is required. Even though there is a similarity (i.e., both use graphs) between the two, they are distinctly different. In a state driven system, the control exists at one particular state (i.e., node) and a transition (i.e., edge) indicates the next control state. In an event driven system (e.g., TR+), control is indicated by the edges, and all events (i.e., nodes) are monitored continuously and can occur simultaneously.

<sup>19</sup>This is when the robot is near the boundary of the current local extent of the potential field.

<sup>20</sup>The local map consists of the CAD map and the previously sensed features that are at least partially located within the new border, or in the node of the abstract global map in which the robot currently finds itself in.



- (vi) If the *global path planner* is *blocked* (i.e., not able to find a path to the goal), put SPOTT in the *stop* state.
- (vii) If the *actuator controller* has not moved the robot in a specified period of time<sup>21</sup>, then notify the *global path planner* that the robot is *blocked* at its current location<sup>22</sup>. The *global path planner* stores this information as a constraint (see Section 9.2 in Chapter 4), so that if the robot is in this position again (i.e., when the global path is replanned), this local route is not used for subsequent paths.

When the robot is started by the operator pushing the *start* button in the *Task Command* section of the *GUI*, the *GUI* “events loop” enters the “*robot is started: initialization*” state for a single iteration, in which the following operations occur:

- (i) A signal is sent to the *global path planner* to clear all stored constraints which indicate where the robot was previously blocked.
- (ii) Place the *robot map* display in local mode. Only the features stored in the map database that are within the local extent of the potential field’s bounds are displayed.
- (iii) If the robot is to be localized before starting execution<sup>23</sup>, then localize the robot by correlating sonar data with the known CAD map (Mackenzie & Dudek, 1994).
- (iv) Start the *actuator controller* process, which in turn performs steepest gradient descent on the potential function, in order to generate robot trajectory commands.
- (v) Start the *TR+ interpreter*, which in turn interprets a TR+ program, and signals back to the GUI controller when the task command has been completed.

The “*robot is stopped: initialization*” state may be entered from the “*robot is executing*” state by either the operator pressing the *stop* button, the TR+ controller notifying the GUI controller that the task command has been achieved, or the global path planner determining that there is no global path to the goal (e.g., doors are closed). The following operations are performed in the “*robot is stopped: initialization*” state before the control enters the “*robot is stopped*” state:

- (i) Notify the *actuator controller* to stop moving the robot.
- (ii) If the *stop* button was pressed, then inform the *TR+ interpreter* to stop interpreting and executing the TR+ program.

<sup>21</sup>Five seconds was found to suffice for a thirty-five by thirty-five square potential function grid. The time is selected so as to give the potential function more than ample time to converge to its solution. If the robot does not move after this period, this indicates that the robot is *blocked* and is not able to achieve its goal.

<sup>22</sup>In addition to the Cartesian coordinates of the robot, the node in the abstract graph map is specified, in addition to the next node in the current global path, as well as the edge that is to be traversed to get to the next node.

<sup>23</sup>An option specifiable by the user.

- ▼ (iii) If the *TR+* interpreter has satisfied the task command (i.e., the top node in the  
 ▲ main program of the currently executing TR+ program), then clear the current task command goal.

If the “*SPOTT system is stopped*” is entered by the user pressing the “*stop SPOTT*” button, then all the processes that are part of the SPOTT system are stopped and killed. Subsequently, the SPOTT program exits.

In addition to controlling the GUI, the *GUI* process is also responsible for informing all other processes of the current operating state, which is defined by the current state of the physical robot. The *GUI* process is an excellent candidate for this role because the user interacts through the GUI (i.e., controlled by the *GUI* process) and the user is one of the main sources for defining transitions between the states. Even though each module (e.g., *TR+* interpreter) is independent to a certain degree, each module’s execution is slightly dependent on the states defined by the *GUI* process.

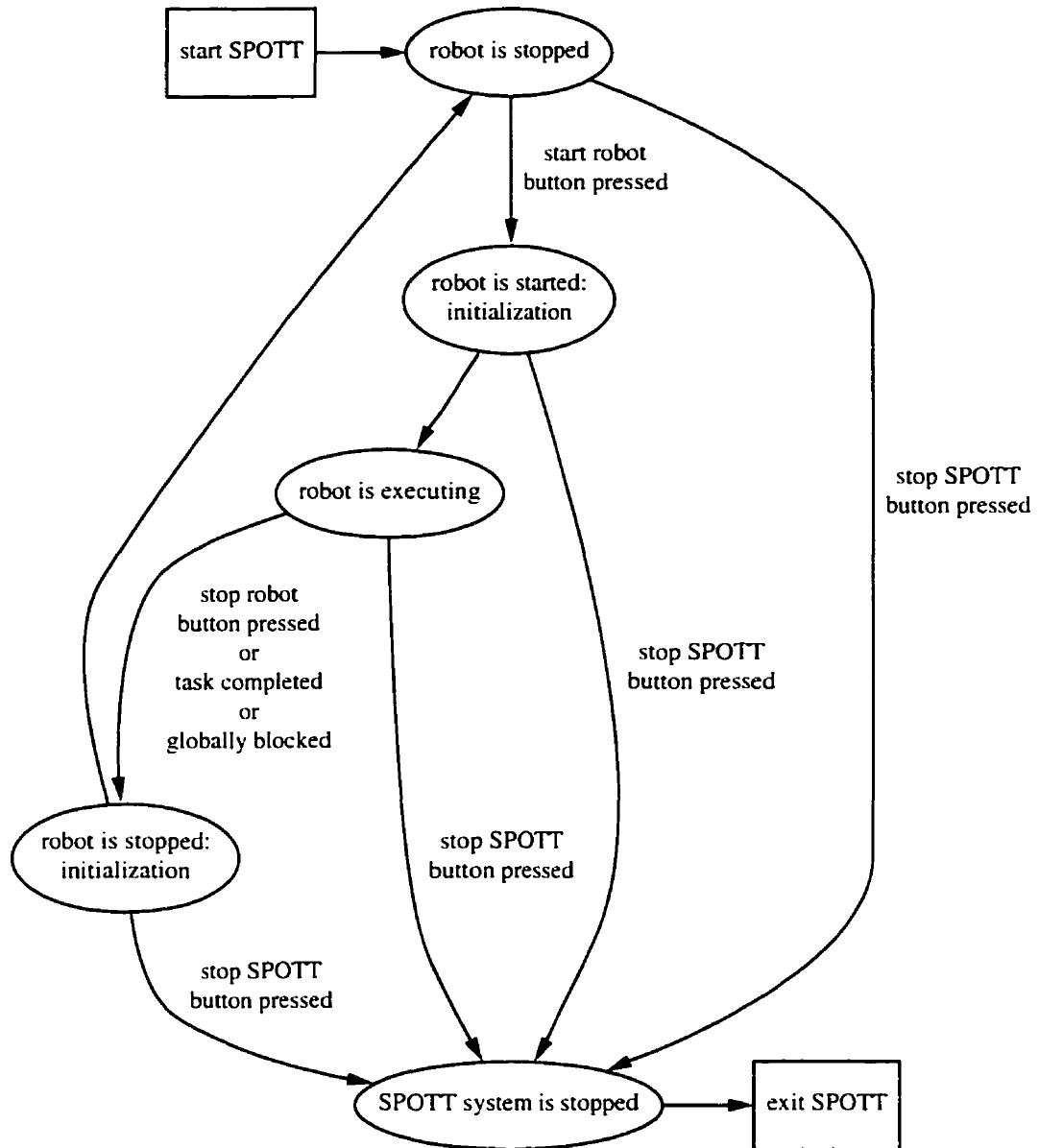


FIGURE 6.4. SPOTT's GUI's State Transition Diagram. Within the graphical event loop there are five main modes of operation : (1) the robot is stopped, (2) the transition from stop to start involving initialization, (3) the robot is executing (i.e., started), (4) the transition from the executing to the stopped mode, and (5) the SPOTT system is shut down. The user may start or stop the robot in all states. During execution, the user may not specify a new task or operational parameters.

### 2.3. Local Path Planning

The potential function (i.e., harmonic function, solution to Laplace's equation) is computed by a collection of processes in a master-slave relationship. The master process also operates as a server process. It handles updates and requests for data on a first-come, first-serve basis. The *GUI* and the *TR+ Interpreter* are the processes which update the potential field by sending updates to the master process. Updates include adding and deleting obstacles or goals, as well as changing an obstacle to a goal, or a goal into an obstacle. The *actuator controller* process requests gradient information at the current robot position estimate which in turn is used to specify the local trajectory commands for the robot. The *GUI*, *TR+ interpreter*, and *actuator controller* processes are also able to send updates to the master process, indicating a new position for the robot. Concurrently during the polling for potential updates from other SPOTT processes, the computation of the harmonic function is performed using a master-slave configuration (see Figure 6.5).

The master process contains the entire discretized grid array of the potential function and sends overlapping subgrids (i.e. a subset of the data) to slave processes to compute a single iteration. After completion of one iteration by a slave process of its data subset, the data are returned to the master process. This sequence of iterations continues indefinitely. New obstacle and goal configurations can be specified concurrently. The master process either contains the current harmonic function or a function which is converging towards the harmonic function for the current configuration. Figure 6.5 illustrates how the grid is divided into four overlapping subgrids. These are required so that new data values from bordering subgrids can be integrated. The original grid is bordered by a one pixel-wide layer to store the boundary-value conditions<sup>24</sup>.

The iterative step of summing the neighbouring elements to obtain the new estimate of the point on the grid can be equivalently solved by a resistive grid (Doyle & Snell, 1984) (see Section 4.1 in Chapter 4). A resistive grid equivalent to the iteration kernel is derived by using Kirchoff's Law for current flowing into a node, and Ohm's Law for a resistor. See figure 4.7 for a resistive grid implementation of the five-point kernel. The nine-point kernel can be similarly represented as a resistive lattice, although diagonal resistors need to be added and the resistor values have to be reportioned accordingly. The potential for implementing harmonic function computation in hardware, providing that the technological

<sup>24</sup>The boundary of the discretized grid is always set using Dirichlet boundary conditions (i.e., saturated high, a source for a flow). However, the boundary conditions for obstacle boundaries can be either Dirichlet or Neumann.

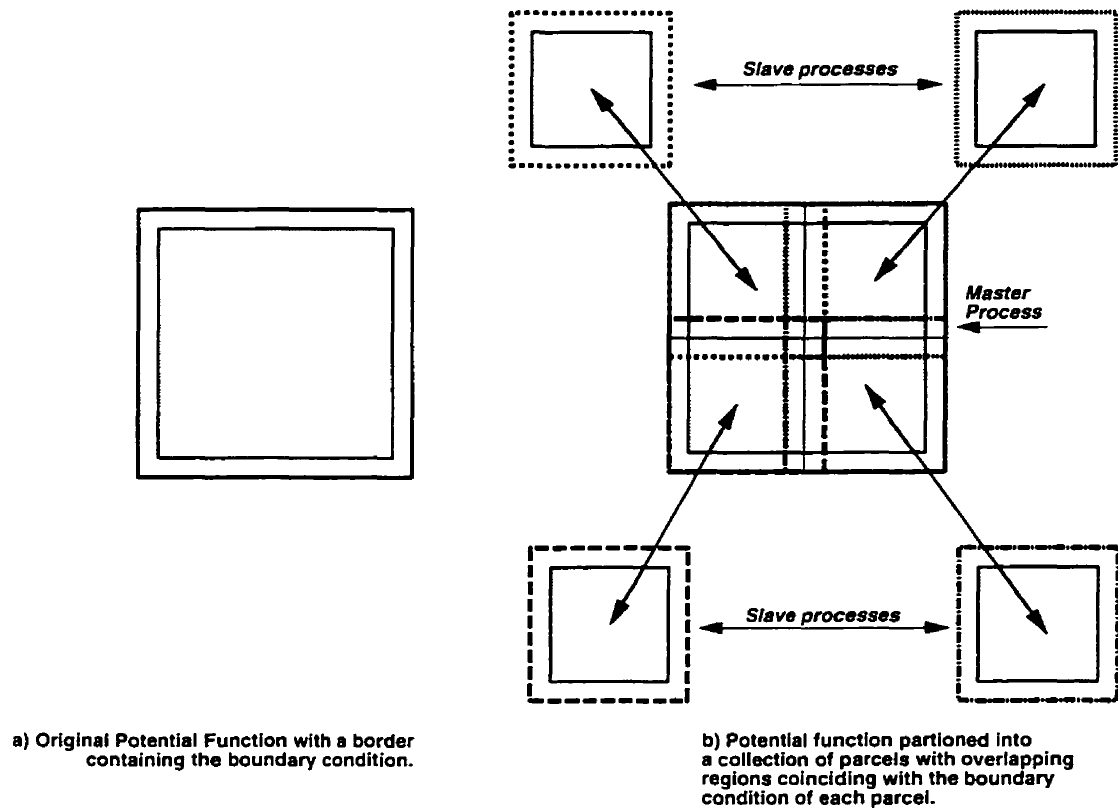


FIGURE 6.5. Potential Field Master-Slave Configuration. The potential field is computed as a server process. The server process responds to updates and requests and concurrently computes the potential field in a master-slave configuration by partitioning the data into sub processes in order to compute single iteration steps.

limitations are overcome, confirms using harmonic functions for path planning in real-time as a viable option.

## 2.4. Map Database

The database consists of the architectural CAD map, the sensed data, as well as an abstraction of the CAD map. The CAD map and the newly sensed features (i.e., line segments) are stored as a symbolic list of geometric descriptions<sup>25</sup> positioned in a Cartesian coordinate space. The local path planner also has a corresponding occupancy grid map (Moravec, 1988). It is a discretized grid, in which grid elements that are occupied by an obstacle are saturated high, and those that are occupied by a goal are saturated low. The free space (i.e., the exclusion of obstacle and goal grid elements) is where the harmonic function is solved. The CAD map abstraction is such that rooms and hallway portions are represented by nodes, and access ways (i.e., doors) are represented by edges.

At this point all incoming sensed data are compared to the existing data (i.e., CAD map and locally sensed data); if they do not already exist, they are added. In addition, the QUADRIS (Bui, in preparation) system's ability to assign labels (e.g., walls, doors) to certain features (e.g., line segments) is made use of when entering QUADRIS features into the map database: if the newly sensed QUADRIS feature corresponds (i.e., spatially) to an existing unlabelled feature already in the map database, then that feature is labelled accordingly.

At first glance, it seems appropriate to use a shared memory model to implement a map database for SPOTT. However, this is not the approach taken for two reasons: (1) SPOTT has been implemented using PVM, and shared memory capabilities within PVM have only been recently introduced<sup>26</sup> (Geist *et al.*, 1994); and (2) the potential field processes require access to the newly sensed data as soon as it becomes available<sup>27</sup>.

Rather than having a single copy of the database, SPOTT has multiple copies, duplicated for the following processes: the *GUI*, the *potential field* master process, the *TR+ interpreter* process, and the *global path planner* process (see Figure 6.6). Not all parts of the database are duplicated for each of the processes. All of the cited processes contain information about the robot's current position and the current task command. Only the *GUI* contains the CAD map, newly sensed features, and the abstract graph map. The *global path planner* process only contains the abstract graph map. The *local path planner* (i.e.,

<sup>25</sup>Currently, the only geometric descriptions used are line segments.

<sup>26</sup>The current version of PVM has shared memory ports which act as a transport mechanism for communication between tasks. If the shared memory communication method is faster than socket calls, it may be ideal for communication with a process that is responsible for maintaining and updating the database. This will require rethinking how the map database functions within the SPOTT architecture and how it should be implemented.

<sup>27</sup>This is because the local path planning module is in a real-time feedback loop with the robot and its environment.

master process), as well as the *TR+ interpreter* process, contain a local window into the set of CAD map and newly sensed features dictated by the local extent of the potential field.

The implementation of the map database can be problematic because each process can change its own copy. All processes that update a portion of the map database are responsible for communicating their changes to all other processes which rely on this part of the database<sup>28</sup>. The difficulty arises when trying to assure uniformity across the multiple copies. To maintain consistency, only one process can update a portion of the map database during any particular state (i.e., robot running or stopped). When the robot is not executing a task, only the *GUI* process can make changes to the map database. However, during execution, different processes can update independent portions of the map database: the *TR+ interpreter* adds newly sensed information; the *actuator controller* process updates the position of the robot; the *GUI* process examines the newly sensed data to determine<sup>29</sup> whether the doors in the abstract graph are open or closed; and subsequently the *global path planner* process can update the global path. The *GUI* process is also responsible for sliding the bounds of the potential field and thus changing the local window within the global map. At this time, all updates to the local map (i.e., newly sensed data) are temporarily suspended (i.e., for a few milliseconds) until the local map has been created with features from the CAD map and previously sensed data that fall within the new local bounds. The temporary blocking of map updates is necessary to ensure so that the local map is consistent across the various processes.

Ideally, it would be desirable to have one copy of the map database (i.e., shared memory). Such an implementation could take on the structure of a blackboard (Carver & Lesser, 1992). The latter is a type of global database which is shared by all modules (i.e., for SPOTT: the *TR+ interpreter*, the *local path planner*, the *global path planner*) of the system. The blackboard contains data and hypotheses (i.e., potential partial solutions), and usually performs problem solving by using an *incremental hypothesize and test* strategy. SPOTT's map database is currently only a data depository. However, future expansions do not exclude the possibility of making it an intelligent (i.e., automatically abstracting and merging sensed data features as they arrive) database. In order to change SPOTT's map database into a blackboard structure, reliable and fast shared memory access is required

<sup>28</sup>The *GUI* process is the only one that relies on all of the entities in the map database.

<sup>29</sup>The sonar and QUADRIS data are correlated with the known door positions (i.e., stored in the abstract graph map) to determine whether a door is open or closed. To determine if a door is open, none of the recently sensed data should correlate with the position of a door contained in the local map provided that the robot is within sensing range of a door. The door can be sensed using sonar or QUADRIS if the robot is within 2 m.

and the issue of quickly updating the potential field's occupancy grid (i.e., via a blackboard structure) to maintain real-time performance needs to be addressed.

Future modifications to SPOTT should address the issue of continual maintenance (e.g., fusing map features, abstracting new nodes from sensor data) for the map database. This will require some reasoning capabilities which may either be part of the map database as a blackboard structure, or determined by an external reasoning agent (e.g., COCOLOG (Caines & Wang, 1995)).



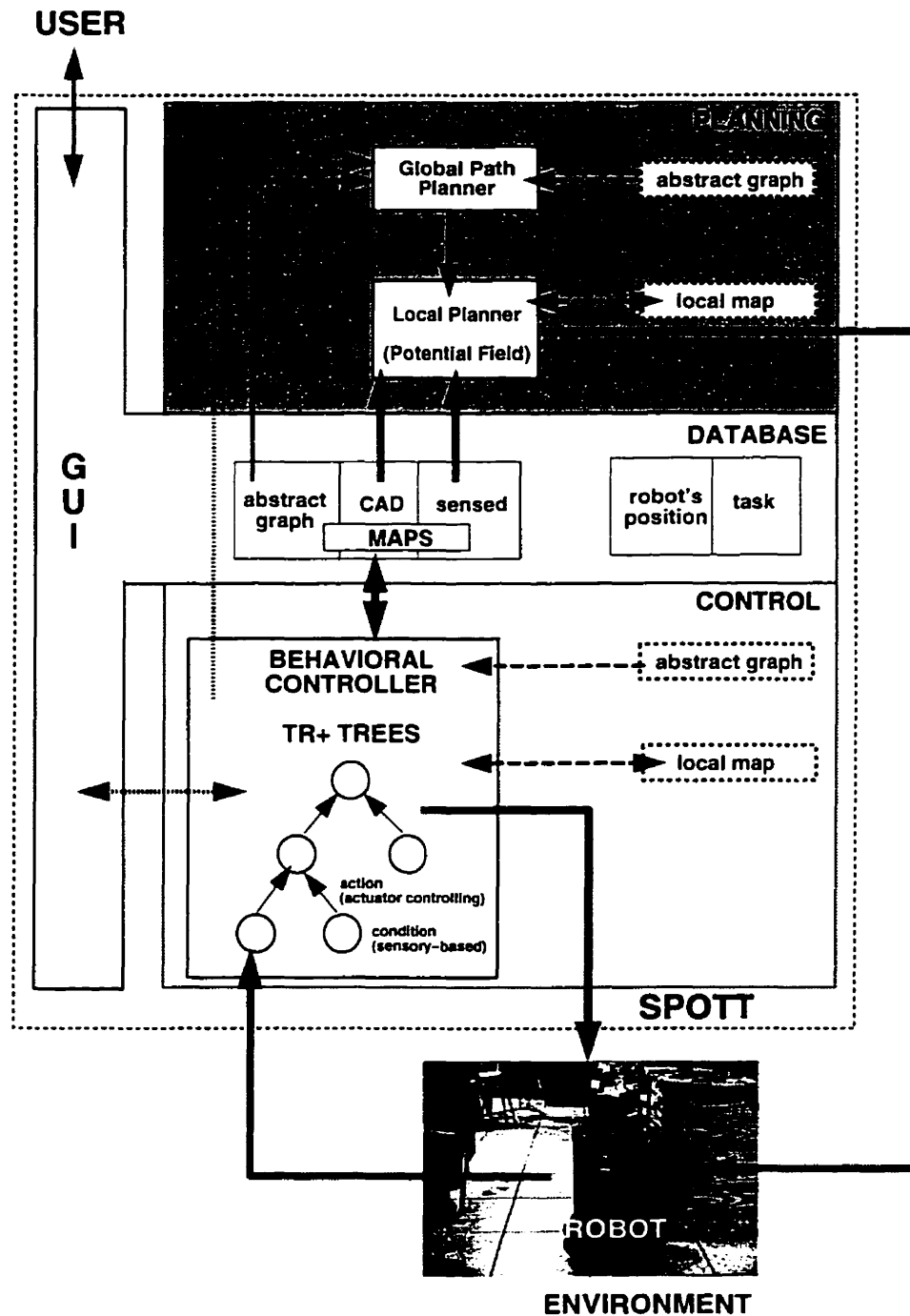


FIGURE 6.6. SPOTT and the Implementation of the Map Database. The entire map database is maintained by the process that controls the *GUI*. The *TR+ Interpreter* and *local path planner* (i.e., master process) have a local map, which is a local window into the set of CAD and newly sensed features. This corresponds to the local window of the potential field of the *local path planner*. The *TR+ Interpreter* and *global path planner* both have a copy of the abstract graph which is only used for reading purposes, and is updated by the *GUI* process. The abstract graph is updated by the *GUI* process when the robot changes position or the target changes. The *TR+ Interpreter* may update the local map, which in turn updates the local map used by the local path planner.

## 2.5. The TR+ Interpreter

The TR+ program interpreter, written in the “C” programming language, is implemented using PVM. Each of the conditions is evaluated concurrently as a separate process. One way to represent TR+ programs is in a tree graph format. The *Dotty* toolkit is used to create software for visually programming TR+ programs as trees, and for monitoring the execution of TR+ programs. The algorithm for the *TR+ interpreter* process is as follows:

- (i) Initialize communication with the *GUI* controller process, and initialize the graphical display, which shows a graphical representation of the currently executing TR+ program.
- (ii) The following sequence of operations are in a continuous loop until the *TR+ interpreter* process is told to stop processing (i.e., when the SPOTT system is shut down by the operator).
  - (a) If the *GUI* controller process sends an update of the local map, then update the local map used by the *TR+ Interpreter* accordingly.
  - (b) If the *actuator controller* process updates the robot position, then update accordingly.
  - (c) If the *GUI* controller process sends a “stop SPOTT” signal, then exit the loop, and subsequently exit the program.
  - (d) If the *GUI* process sends a “start execution” command with a particular task command, then do the following in a continuous loop until told otherwise:
    - (i) If the *GUI* updates the local map, then update accordingly.
    - (ii) If the user (i.e., via the *GUI*) stops the robot, then kill all condition processes currently executing and exit this loop.
    - (iii) If the *actuator controller* process updates the robot position, then update accordingly.
    - (iv) Execute the TR+ program:
      - (A) *The main TR program’s conditions are executed (i.e., processes are spawned) and initialized.*
      - (B) *The highest level TRUE condition specifies which action  $a_i$  is activated. If  $a_i$  is a primitive action, then it is active till overridden. If  $a_i$  is a subroutine TR program, then its conditions are executed (i.e., processes are spawned) and initialized (if they have not been as of yet), and (ii) is repeated for this particular subroutine TR program.*

- (C) *If a subroutine TR program has not been called for a redefined period of time, the execution of its conditions is stopped (i.e., processes are killed).*
- (D) *If the highest level condition in the main TR program is TRUE, then the interpreter STOPS, otherwise execution returns to step (ii), using the main TR program.*

Currently, when the *TR+* interpreter starts interpreting and executing a *TR+* program, all the condition processes are spawned before they can be used for computation. The startup of a process is a very time consuming operation<sup>30</sup>. A future modification to the *TR+* interpreter would be to have it scan the *TR+* program before run-time for all conditions which contain processes that need to be spawned. These condition processes would be in a doormat state until activated by the *TR+* interpreter. This would alleviate the lag currently experienced in the start up of a *TR+* program. This is especially evident for task commands that require very little time for execution (i.e., when the target location is close to the starting position of the robot).

In addition to a graphical display for visualizing the state of the currently executing *TR+* program, a graphical tool for programming *TR+* programs has also been written.

#### 2.5.1. Programming *TR+* Programs

A graphical editor (i.e., creator) of *TR+* programs, called “**TR+ edit**”, has been implemented using *Dotty*. A menu-based graphical interface for specifying attributes of the conditions, actions and *TR+* programs was also integrated within the graphical editor (see Figure 6.7).

Before using the graphical editor to program the *TR+* program, a collection of conditions, actions, and variables (i.e., primitive building blocks) must be defined and created. These are the elements which are manipulated to create the *TR+* program. They are redefined in a set of ASCII files (i.e., written in the C language, in .c and .h files), and are the entities (i.e., building blocks) of a *TR+* program. In addition to the files containing the definitions of the entities, there is a collection of programs which compile into executables for performing the computation of the conditions. The model used is that each condition process accesses a process (e.g., *robodaemon*) which provides it with sensor data (i.e., processed or raw). Actions are performed as part of the *TR+* interpreter cycle. See Appendix B for a list of conditions and actions currently implemented by the SPOTT system.

<sup>30</sup>It varies by machine type and the current resource allocations on the machine. Sometimes it can take several seconds.

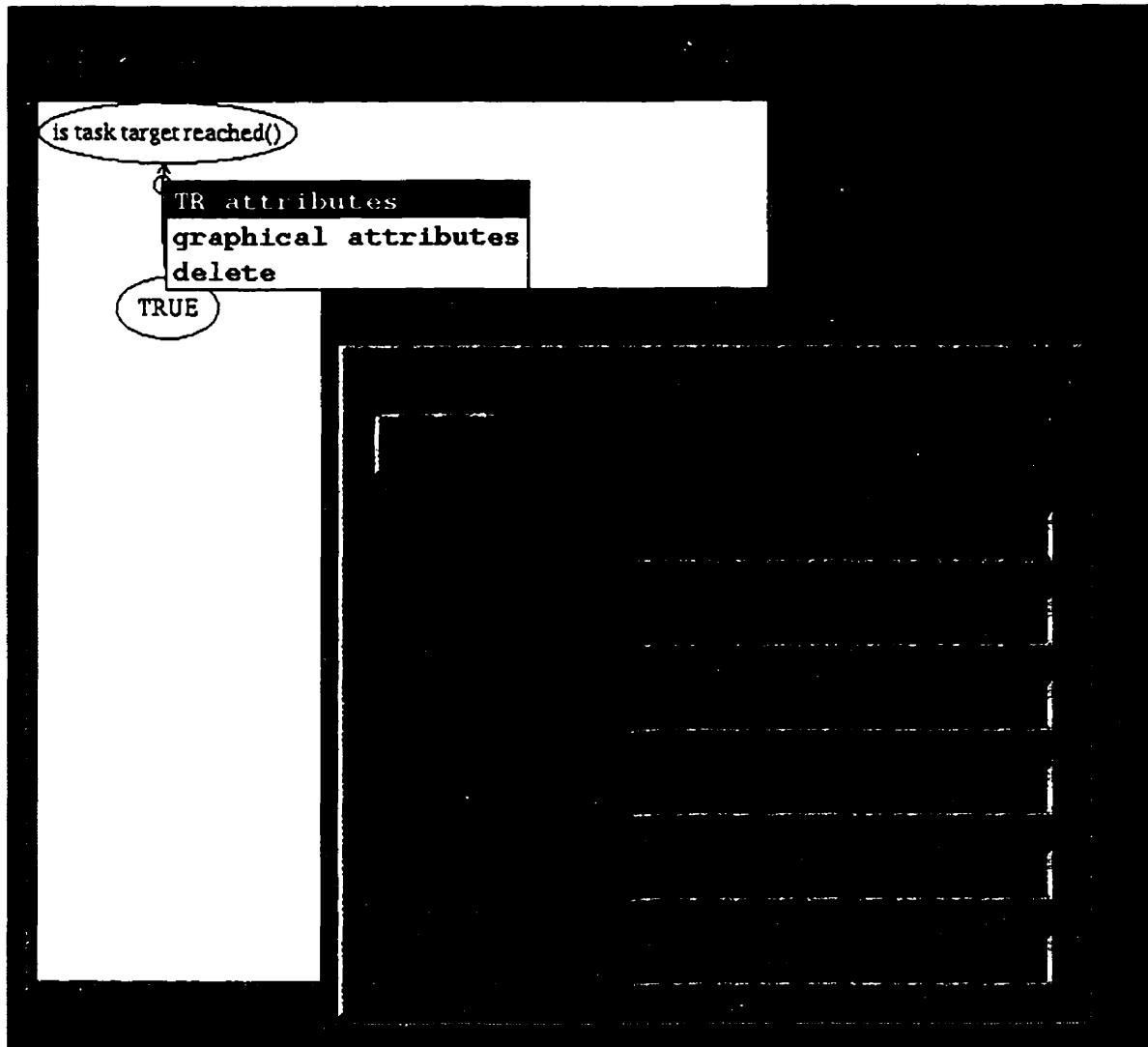


FIGURE 6.7. **TR+ Programming With TR+edit.** A graphical editor (i.e., creator) of TR+ programs, called "TR+ edit", has been implemented using *Dotty*. A menu-based graphical interface for specifying attributes of the conditions, actions and TR+ programs was also integrated within the graphical editor. Illustrated is the graphical editor and a menu for editing the attributes of the selected arc. Nodes are created by clicking on the display space with the left mouse button. Arcs are created by clicking with the second mouse button on the source node and holding the mouse button down until the cursor is over the destination node. The third mouse button, when the cursor is placed over a node or arc, is used to pop up a menu which permits the selection of the node's or arc's attributes.

## 2.6. RoboDaemon

*Robodaemon* is a software development system which builds an abstraction of a robotic environment (Dudek & Jenkin, 1993) that permits external software to interact with either a simulated robot and environment or to a real robot complete with sensors (i.e., transparent). It permits processors to communicate from anywhere on the net through a socket and

machine address. It also provides a realistic simulator of 2D time-of-flight sonar sensing (Dudek *et al.*, 1993), which can be used for simulating sonar sensing. This is useful for the conditions in a TR+ program that use sonar as input. *robodaemon* has its own graphical user interface, which can be used for controlling the robot and testing the robot's functionality. A disadvantage of *robodaemon*'s socket-based communications protocol is "*that it imposes a data-dependent communications delay on interactions*" (Dudek & Jenkin, 1993). Since most of the sensor information, and control information has to flow through *robodaemon*, this interface can be a potential bottleneck, especially if it blocks communication to other processes. This is the case when the sonar sensors are acquiring data<sup>31</sup> and *robodaemon* does not listen to other commands (i.e., they are discarded). A future modification to the *robodaemon* software package should include a PVM interface. PVM comes equipped with its own error handling and non-blocking calls, and handles the queuing and buffering of message requests.

*Robodaemon* provides a generic interface which supports many different mobile robot platforms. This permits control software (e.g., SPOTT) (and sensor algorithms) to be developed independently of the specific platform. Future modifications to *robodaemon* include the queuing and buffering of message requests to help alleviate potential bottlenecks when communicating to the robot via *robodaemon*.

---

<sup>31</sup>The Nomadics 200 robot has an array of sixteen sonar sensors evenly distributed on the circumference of the upper part of its turret. To acquire a dense scan of the environment, the turret is rotated by appropriate incremental angles.

### 3. Reasoning System Interface

One of the goals of the SPOTT system is to bridge the gap between the low and top levels (i.e., reactive and symbolic control). SPOTT is not a completely distinct layer, but instead integrates behavioral (i.e., reactive) control, planning and some symbolic reasoning into a cohesive system. In essence, SPOTT is a merging of the low and middle layers. It also does some reasoning using the global path planning module. However, the latter could be replaced by a symbolic reasoner in order to perform more than just AI search. Many navigational tasks can be performed with SPOTT, but task completion in certain scenarios will require additional - more complex - reasoning capabilities.

The COCOLOG (Caines & Wang, 1995) reasoning system being developed by Caines and his group at CIM is the one planned to be interfaced with SPOTT. Preliminary integration of COCOLOG with SPOTT has involved COCOLOG replacing the *global path planner* process (see Figure 6.8). The user specifies a task command goal, and then *COCOLOG* takes control of SPOTT by specifying a set of intermediate goals<sup>32</sup>. The user resumes control once COCOLOG has achieved the task command goal. The actual use of COCOLOG in this scenario is not so interesting because COCOLOG's role is redundant (i.e., SPOTT's global path planner can already perform this operation). What is interesting is the integration of SPOTT with COCOLOG. The specification of intermediate goals by COCOLOG will become useful when these goals are part of a complex exploration task (e.g., *find the red ball* or *find the chair*). This is one of the potential roles for COCOLOG.

Other future uses of the COCOLOG reasoning system could include dynamically specifying changes to the TR+ programs during execution, possibly learning new TR+ control programs, learning the abstract graph map in environments where this is not specified beforehand<sup>33</sup>, and map database maintenance (i.e., verifying, refuting, and merging symbolic data constructs). Map database maintenance is either a role for a reasoning agent such as COCOLOG or for a blackboard architecture (see Section 2.4).

---

<sup>32</sup>The intermediate goals are Cartesian coordinate points specifying the centre of edges (i.e., doors). COCOLOG sends a procedural list of intermediate goals (i.e., SPOTT processes each one of these to completion in sequential order) which are at the locations of edges that are to be traversed.

<sup>33</sup>In the current implementation of SPOTT, the user creates the abstract graph map based on an architectural CAD map. It would be ideal to automate this process before execution, or to learn this abstract graph while building a map.

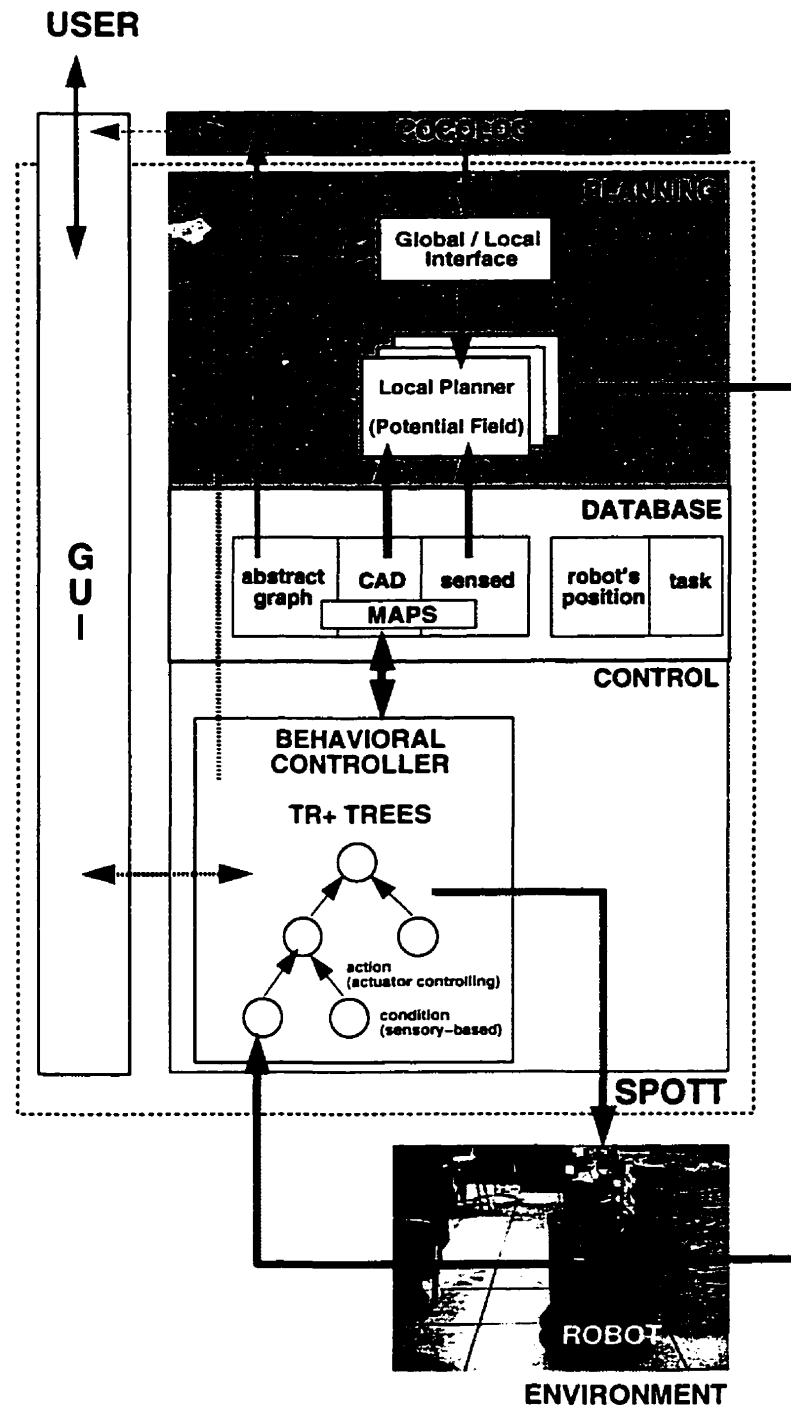


FIGURE 6.8. Integration of COCOLOG with SPOTT. In the initial integration, COCOLOG performs the global path planning and provides SPOTT with a procedural list of intermediate goals. SPOTT notifies COCOLOG after the successful completion of each of the intermediate goals and when a goal is not achievable (e.g., doorway was blocked).

#### 4. Modularity and Portability

SPOTT's implementation uses a collection of communication processes executing across a heterogeneous network of processing resources. This implementation is modular and easier to understand conceptually because conceptual modules correspond to an actual set of computing processes. The heterogeneity property of PVM makes using different architectures (e.g., SUN, SGI) relatively transparent to SPOTT's software. There is always the potential of executing the entire SPOTT system on board the robot if sufficient processing resources were made available. The portability of the SPOTT system, chiefly due to its implementation using PVM, makes it truly hardware independent. This permits taking advantage of faster processors as they become available. The ability to run UNIX (i.e., PVM is chiefly designed to run within a UNIX environment) on personal computers (i.e., using Linux) opens up an opportunity to use a relatively inexpensive hardware platform for executing SPOTT.



## CHAPTER 7

---

### Experiments

SPOTT has been implemented as a distributed control system chiefly because of the computational requirements of the local path planner, and the conditions associated with a TR+ program. The local path planner (i.e., potential field) computes the solution to Laplace's equation (i.e., used as the potential function) using an iterative (i.e., parallel) computation. It is only natural<sup>1</sup> to also concurrently compute each one of the TR+ conditions as separate processes.

SPOTT's performance not only depends on the method of implementation<sup>2</sup>, but also on the rate at which the TR+ interpreter and local path planner<sup>3</sup> are updated by sensor information and consequently move the robot. Other measures of performance include robustness, reliability and whether the task was successfully completed. Five sets of experiments were performed which looked into different performance aspects of the SPOTT system:

- (i) The first experiment was a simple test of the communication speed on the network at CIM. Messages were passed back and forth between processes in different configurations (see Section 1).
- (ii) The second set of experiments looked into speeding up the performance of the local path planner (i.e., computing the harmonic function as the potential function) (see Section 2).
- (iii) The third set of experiments investigated the quality of the sensor (e.g., sonar. QUADRIS) data and the performance of the mapping and localization algorithms which rely on this data (see Section 3).

---

<sup>1</sup>In theory, the computation of each of the TR conditions is done in parallel.

<sup>2</sup>The method of implementation uses parallel processes and a message-passing paradigm.

<sup>3</sup>Both are in a real-time feedback loop with the robot and the environment.

- (iv) The fourth set of experiments involved using the entire SPOTT system. Two navigational tasks consisting of maneuvering to a known spatial location were executed. In the first navigational task, there was no a priori map, while in the second there was.
- (v) Finally, the last set of experiments was conducted at the 1996 IRIS-PRECARN conference at the Queen Elizabeth Hotel in Montreal, between the dates of June 4-th through to the 6-th, 1996. Three operator consoles as well as the mobile robot were taken to the Queen Elizabeth Hotel and the control system was executed on these workstations as well as on a collection of workstations back at McGill University. The mode of communication between the Queen Elizabeth Hotel and McGill University was Ethernet over an optical fibre link.

In addition to evaluating whether SPOTT can successfully achieve a navigational task (iv, v), the other experiments (i, ii, iii) evaluate the following important performance factors:

- (i) The communication bandwidth of the message passing system (i.e., PVM) provides an upper limit for how fast SPOTT can perform. This is especially true for the local path planner whose performance depends on the rate at which data is passed between the master and slave processes.
- (ii) The performance of the local path planner is a key element because it links perception with action (i.e., trajectory planning). It is important that its response time to environmental changes be minimized as much as possible.
- (iii) Finally, SPOTT's ability to react to its environment is constrained by how well (i.e., and how fast) it can see (i.e., sense and perceive) the world.

## 1. Message-Passing Costs on a Network

SPOTT's distributed control network is implemented using the message-passing software package called PVM (Parallel Virtual Machine). When using PVM, the user specifies a collection of workstations<sup>4</sup> upon which the applications will be executed, and PVM decides where the processes get executed. Since SPOTT is used to control a robot in real-time<sup>5</sup>, it is desirable to choose the fastest machines for PVM (i.e., from the set of accessible machines). This ensures that the individual processes will be executed as quickly as possible, but the execution speed of an application parallelized across a message-passing paradigm is also dependent on the inter-process communication times.

The network at CIM is interconnected using the Ethernet protocol. Some of the workstations are connected in local clusters. Table 7.1 lists the average (i.e., over a run of 20 iterations) inter-process communication times for peak and low periods between processes in different configurations. Each test consisted of initiating communications by sending the smallest possible packet (i.e., four bytes) for 20 iterations, and increasing the package size every subsequent 20 iterations. Table 7.1 shows that the time taken to pack the data is negligible and it is the communication (i.e., *send*) time which is the limiting factor for performance in a message passing paradigm. The initial communication with another process experiences some startup delays. As expected, communications between processes on the same workstations are the fastest (i.e., see the *send* column in Table 7.1). However, if both of these processes are heavily dependent on computational resources, they compete against each other. Next fastest are communications between processes on the same local cluster, about 40% faster than communications between processes on the general network. It would be interesting to see whether the performance would improve linearly if the network were linked to an ATM or "Fast Ethernet" protocol<sup>6</sup>.

Sometimes, there is significant fluctuation (i.e., outliers) among the twenty iterations for each of the inter-process bandwidth communication tests using PVM (see (b), (c), and (e) in Figure 7.1). In addition, the first attempt at communication takes a lot longer than subsequent exchanges<sup>7</sup>. This needs to be taken into consideration, especially in processes that are relied upon for real-time performance. For example, SPOTT's local path planner continually iterates in a master-slave configuration (see Section 2). It was found that this

<sup>4</sup>The collection of workstations to run PVM on can be chosen from the local network, and can even include other workstations accessible via the internet. Communication to machines outside the local network is usually slower. PVM executes on a subset of machines accessible to the user.

<sup>5</sup>Real-time means that SPOTT responds faster than the time taken for changes in the environment.

<sup>6</sup>*Fast Ethernet* has a bandwidth of 100 M bps, while Ethernet has a bandwidth of 10 M bps.

<sup>7</sup>See the rows in Table 7.1 for the data packet size of 4 bytes. See also (a) in Figure 7.1

Data Packet Size (bytes)	<i>same machine</i> Stargazer		<i>local cluster</i> Voyager		<i>local cluster</i> Defiant		<i>network</i> Vangogh	
	Pack ( $\mu$ sec)	Send ( $\mu$ sec)	Pack ( $\mu$ sec)	Send ( $\mu$ sec)	Pack ( $\mu$ sec)	Send ( $\mu$ sec)	Pack ( $\mu$ sec)	Send ( $\mu$ sec)
	peak							
4	-	<b>25052</b>	-	<b>11763</b>	-	<b>28175</b>	-	<b>83885</b>
100	136	<b>3940</b>	153	<b>5897</b>	142	<b>6548</b>	109	<b>9705</b>
1000	169	<b>4101</b>	185	<b>7105</b>	187	<b>7586</b>	127	<b>12526</b>
10000	741	<b>8063</b>	727	<b>20025</b>	740	<b>22282</b>	451	<b>35843</b>
100000	5504	<b>54546</b>	5493	<b>335595</b>	5445	<b>174660</b>	5999	<b>273168</b>
1000000	51404	<b>533424</b>	52975	<b>1531164</b>	50985	<b>1533180</b>	57120	<b>2519459</b>
	low							
4	-	<b>40515</b>	-	<b>11307</b>	-	<b>27796</b>	-	<b>75259</b>
100	141	<b>3978</b>	133	<b>5816</b>	142	<b>6443</b>	166	<b>13492</b>
1000	169	<b>4128</b>	173	<b>6774</b>	181	<b>7511</b>	182	<b>12568</b>
10000	764	<b>7604</b>	736	<b>22183</b>	733	<b>21697</b>	497	<b>36243</b>
100000	5692	<b>64506</b>	5420	<b>161940</b>	5421	<b>158505</b>	6795	<b>261589</b>
1000000	55956	<b>525037</b>	51061	<b>1518712</b>	51146	<b>1516647</b>	55042	<b>2534524</b>

TABLE 7.1. PVM Bandwidth Tests At Peak and Low Periods. Messages were sent from a process on the SGI Indy workstation *Stargazer* to a process on the same machine, to a process on a SGI Indy workstation *Voyager* which is on the same local cluster, to another process on a 586 PC computer *Defiant* on the same local cluster, and to another process on a SUN4 workstation *Vangogh* on the same local area network but not on the local cluster. Each test consisted of initiating communications by sending the smallest possible packet (i.e., four bytes) for 20 iterations, and subsequently increasing the package size every 20 iterations, sending the same size packet for 20 iterations. The average times for the packing of the data and transmission over the 20 iterations are shown above. The *packing* time is negligible compared to the *sending* time, and therefore it is the *send* time which is the discriminating parameter between the different arrangements.

computation required at start up that the first two iterations be synchronized. If this were not the case, some of the slave processes would race ahead and process their parcel of data a lot quicker than the other processes. After the first two synchronous iterations, the master and the set of slave processes continue in an asynchronous fashion.

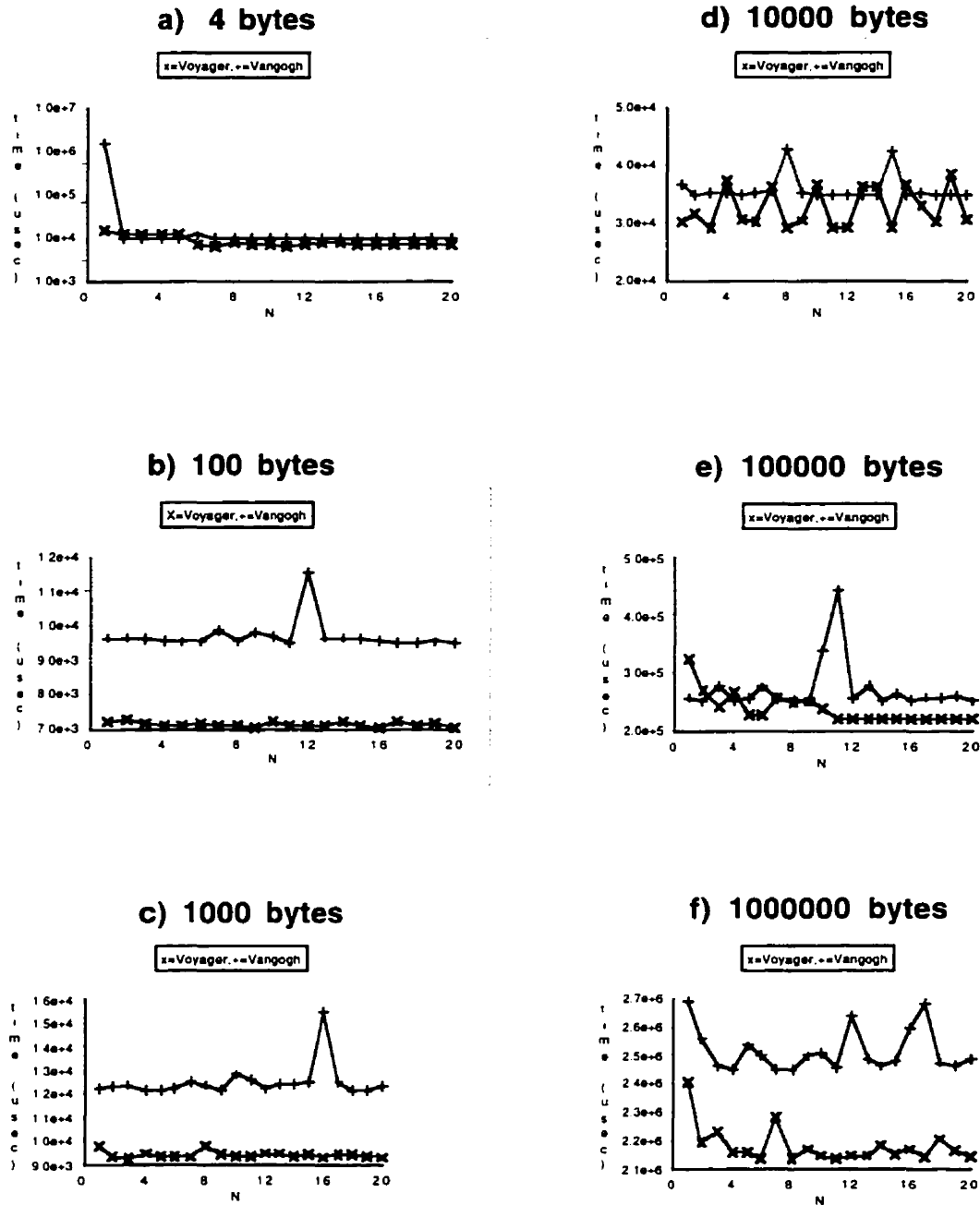


FIGURE 7.1. PVM Bandwidth Fluctuation Tests. The test results documented in Table 7.1 are the averages over 20 consecutive transmissions. The above figure shows the peak transmission rates for the 20 tests between *Stargazer* and *Voyager*, and between *Stargazer* and *Vangogh* workstations for different data sizes. *Stargazer* and *Voyager* are both SGI Indy workstations on the same local cluster, while *Vangogh* is a SUN4 workstation on the same network. The network is a multi-user network, so fluctuations in data transmission will occur depending on what other users are concurrently executing on the network.

Table 7.1 shows the average communication times for different packet sizes. Most of the data packets sent between SPOTT's processes are less than 1000 bytes in size, which translates to each communication taking between 4 to 13 msec<sup>8</sup>. When large chunks of data are sent (e.g., images<sup>9</sup>), communication times need to definitely be considered when looking at system performance. Since SPOTT is executed on a multi-user network, there are going to be fluctuations in the data rate over time because there is no way to anticipate and predict all activity on the network. This is a potential downside in executing a system on a multi-user heterogeneous network, but the potential for portability when using PVM outweighs this disadvantage under most circumstances.

---

<sup>8</sup>See the rows in Table 7.1 for the data packet size of 1000 bytes. See also (c) in Figure 7.1

<sup>9</sup>Communication times for sending an image can be interpolated from Table 7.1. Assume that the image is 512 by 512 pixels and that a single byte is used to store each pixel (i.e., 262144 bytes in total). The average time to send such an image between processes on the same workstation is 138 msec, on the same local cluster is 348 msec, and on the general network is 562 msec.

## 2. Local Path Planning

The local path planner computes the harmonic function (i.e., potential function) using an iterative computation. This has been chosen so that a steepest gradient descent value is always available at the corresponding estimated position of the robot.

Performance of an iterative computation is improved by either reducing the time of a single iteration, or the number of iterations required for convergence. The tests to examine single iteration performance were done using Gauss-Seidel iteration with ADI (i.e., Alternating Direction) techniques (see Section 4.1.1 in Chapter 4). The various performance improvements tested are as follows:

- Computing the harmonic function in parallel using PVM results in at least halving the time required for a single iteration as compared with computing the harmonic function using a single process.
- The computation time of the harmonic function is directly proportional to the number of grid elements (see Figure 7.2). Typically SPOTT uses a potential function with a grid size of 35 by 35 points. Using this grid size results in convergence to the harmonic function in less than 2 seconds.
- There should be at least one workstation devoted to each of the processes used in computing the harmonic function in a master-slave configuration. Recall that each slave process iterates over a parcel of the entire data set. The optimum number of slave processes to use is dependent on the size of the entire data set as well as the overhead associated with the packing and sending of the data in a message passing paradigm. Figure 7.3 shows the relationship between the computation times for a single iteration with respect to the number of slave processes used (i.e., empirical results).
- Using the faster machines (e.g., SPARC workstations as opposed to SUN 4 workstations) results in a speed advantage of at least 20%. In addition, using machines that are on the same local cluster also improves performance (see Table 7.1).
- Executing the slave processes asynchronously with the master process results in an average 33% improvement over a synchronous implementation. The master process receives the results of a slave process as they become available. In a synchronous implementation, the master process waits for all the slave results before starting a new iteration. The only problem with using an asynchronous implementation with slave processes is that the first two iterations need to be run synchronously. If this is not done, some of the slave processes hang up on the network, and these processes

fall behind in the number of iterations they execute when compared to the processes that do not hang up<sup>10</sup>. Synchronizing the first two iterations results in the number of iterations of all the slave processes to be within 10% of each other.

The improvements mentioned above make a single iteration faster by at least 73% (i.e., 50%, 20% followed by 33% improvements). The time taken for a single iteration is also dependent on the number of grid points (see Figure 7.2) as well as the number of slave processes (see Figure 7.3).

Other factors that were considered are related to the overall convergence of the harmonic function, as opposed to a single iteration, and are as follows:

- SOR (Successive Over-Relaxation) techniques help (see Section 4.1.1 in Chapter 4) speed up convergence by a factor related to the grid dimension.
- A good first estimate of the unknowns makes it possible for the convergence of the harmonic function to be solved in a relatively small number of iterations (Hornbeck, 1975). The use of the summation of potentials approach (see Section 3.3 in Chapter 4) as an initial solution for the computation of the harmonic function produces a solution which converges in about 20% less iterations. SPOTT typically uses the summation of Gaussian potentials as an initial solution. However, other potential functions and varying parameters associated with the initial resting value of the potential function appear to slightly speed up the convergence rate (see Figure 7.4).

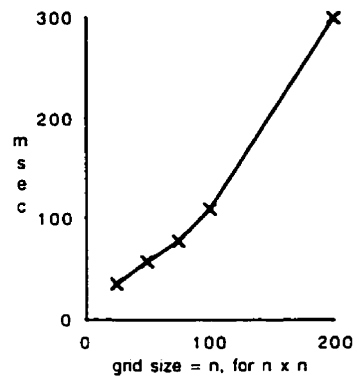
Other factors that did not necessarily speed up the convergence rate or speed up a single iteration, but are more related to the overall functionality of the system are as follows:

- The use of a nine-point kernel (see Section 4.1 in Chapter 4) is more accurate than the five-point kernel, and there is no significant additional computational expense. The nine-point kernel appears to give smoother intermediate results.
- The checking<sup>11</sup> of whether a slave process has sent data to the master process is done by a polling loop in the master process. The order in which slave processes are polled by the master process depends on a hierarchy which is determined by the distance of the goal and robot to the centroid of the parcel of data associated with the slave process. The slave processes whose centroid are closer to the goal and robot positions are processed first. The ordering permits the poll checking of which slave process have computed their iteration to be relevant to the navigation of the robot.

<sup>10</sup>This appears to be an undesirable side effect of how processes are executed and communications are initiated through socket connections. It is not clear if this can be or will be corrected in PVM.

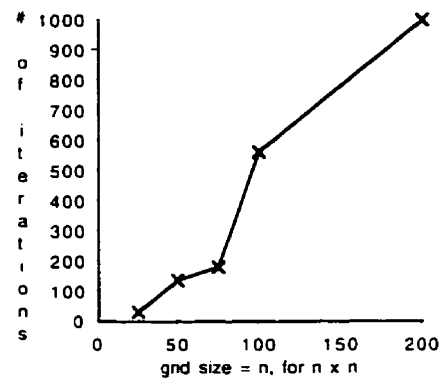
<sup>11</sup>This is done using a non-blocking PVM function.





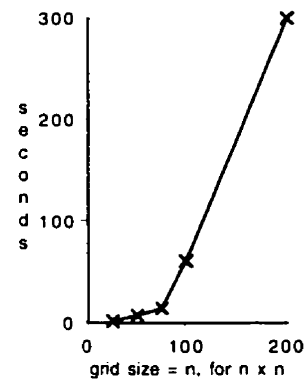
Time taken for each iteration for different grid sizes.

a)



Number of iterations to converge for different grid sizes.

b)



Total time for convergence for different grid sizes.

c)

**FIGURE 7.2. Potential Field Computation Times Versus Grid Size.** (a) The computation time of a single iteration increases proportionately with the number of grid points (i.e., sampling resolution). (b) In addition, the number of iterations required for convergence also increases. (c) The total time for convergence appears to be exponential with respect to the number of grid points. Therefore, it is desirable to keep the number of grid points to a reasonable level to have any resemblance of real-time performance. Most of the experiments with the robot used a grid size of 35 by 35 points.

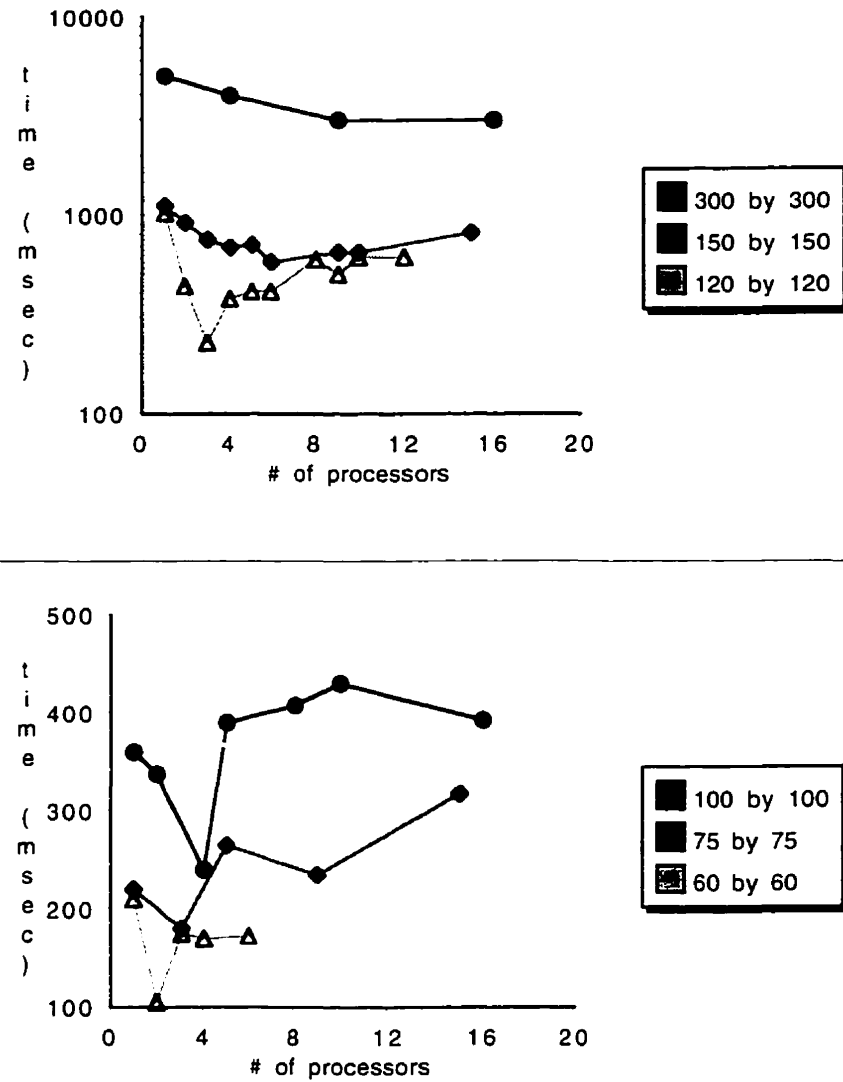


FIGURE 7.3. **Computation Times for a Single Iteration:** The computation time (in milliseconds) of a single iteration for different grid sizes is plotted against the number of slave processes. Each set of data corresponds to a different overall square grid size. Each slave processor processes an equal amount of data. The optimum number of processors to use depends on the overall grid size. The experiments were performed so as to ensure that each master and slave process were executed on a unique processor (i.e., workstation).

The improvements made in order to reduce the time taken for a single iteration as well as to minimize the number of iterations help to speed up the convergence process significantly. Convergence time is also dependent on the speed of the processors as well as the communication protocol (see Section 1).

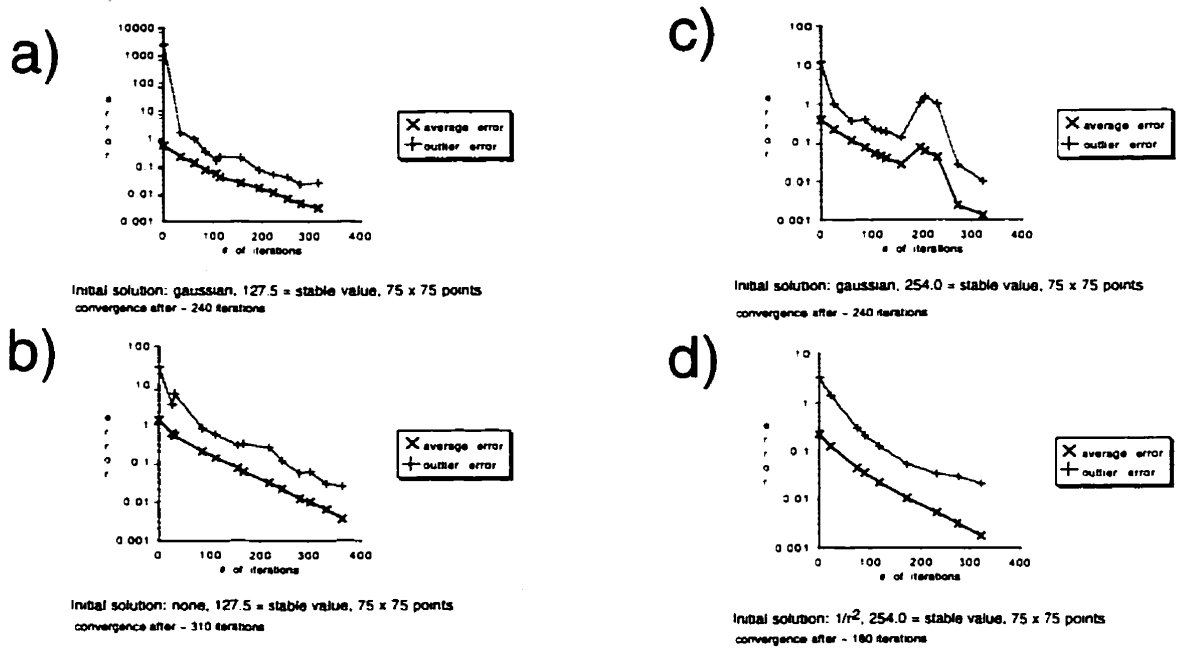


FIGURE 7.4. Relation of Initial Guess to Convergence Rate. The graphs show the difference in value (i.e., of the grid points) between successive iterations (i.e., error) plotted against the number of iterations executed. Shown are both the average over all grid points and the maximum error (i.e., outlier). In all of the illustrated cases shown above, the summation of potentials approach was used as an initial guess to the computation of the harmonic function. The goal saturation value is 0, while the obstacle saturation value is 255. In (a) and (c), Gaussian potentials were used, (b) was processed with no initial solution, and (d) used a  $\frac{1}{r^2}$  potential.

### 3. Perception

#### 3.1. Mapping Using Sonar

The sonar sensor is rated up to an 8 m range (according to Nomadics Technologies); although, multiple reflections may reduce the reliability of the data at this range. In open spaces, such as a laboratory area, the sonar can relatively accurately portray the environment up to the specifications of the sensor (see Figure 7.5). Trouble occurs in corners and narrow corridors (see Figure 7.6), where multiple reflections result. In these cases, the sonar range data does not accurately reflect the environment.

The results in Figures 7.5 and 7.6 have been processed with an algorithm (Mackenzie & Dudek, 1994) that eliminates outliers and fits straight line segments to a cluster of sonar data points. In order to obtain reliable line segments, sonar data points beyond a certain threshold are discarded<sup>12</sup>. For open spaces, this threshold value can approach the range specification of the sensor (see Figure 7.5), while in areas where multiple reflections are possible (e.g., corners, narrow corridors) (see Figure 7.6), this threshold value should be kept to a minimum (e.g., set to 2 m).

Some potential enhancements to the processing of the sonar data which might improve the quality of the data include taking advantage of a priori knowledge (e.g., CAD map) (and possibly other sensor data). For example, in areas where multiple reflections occur (e.g., corners), CAD map information can be used to discard sonar data that do not accurately reflect the environment<sup>13</sup>. This type of improvement will increase the reliability of the sonar data. Since time is an important element for real-time performance, it is necessary to keep the time taken for sonar data collection and clustering into line segments to a minimum. Presently, this process takes approximately 5 seconds to compute, which is too long<sup>14</sup>.

<sup>12</sup>Sonar data is sent out in a cone pattern, and the further away the obstacles are with respect to the robot (i.e., from which the data is reflected back), the less reliable the data is.

<sup>13</sup>The CAD map is used to form a hypothesis of what the sonar sensor should return. This is relatively easy to do in a hallway where there are very few obstacles that are not captured in the CAD map. This is not the case in a room, where there can be a fair amount of clutter. In this case, it would be difficult to determine if the sonar data is sensing an obstacle not in the CAD map or just sensing a corner (i.e., with multiple reflections). Other sensor data (e.g., QUADRIIS) should be used to verify sonar data in rooms.

<sup>14</sup>If the robot is moving at 1 m/sec and the sonar range is 3 m, then the robot will have moved 5 m during the time it takes to process the sonar data. This is assuming that the sonar sensing is concurrently executing while the robot is moving. At this rate, the sonar data will be irrelevant with respect to the current position and heading of the robot.

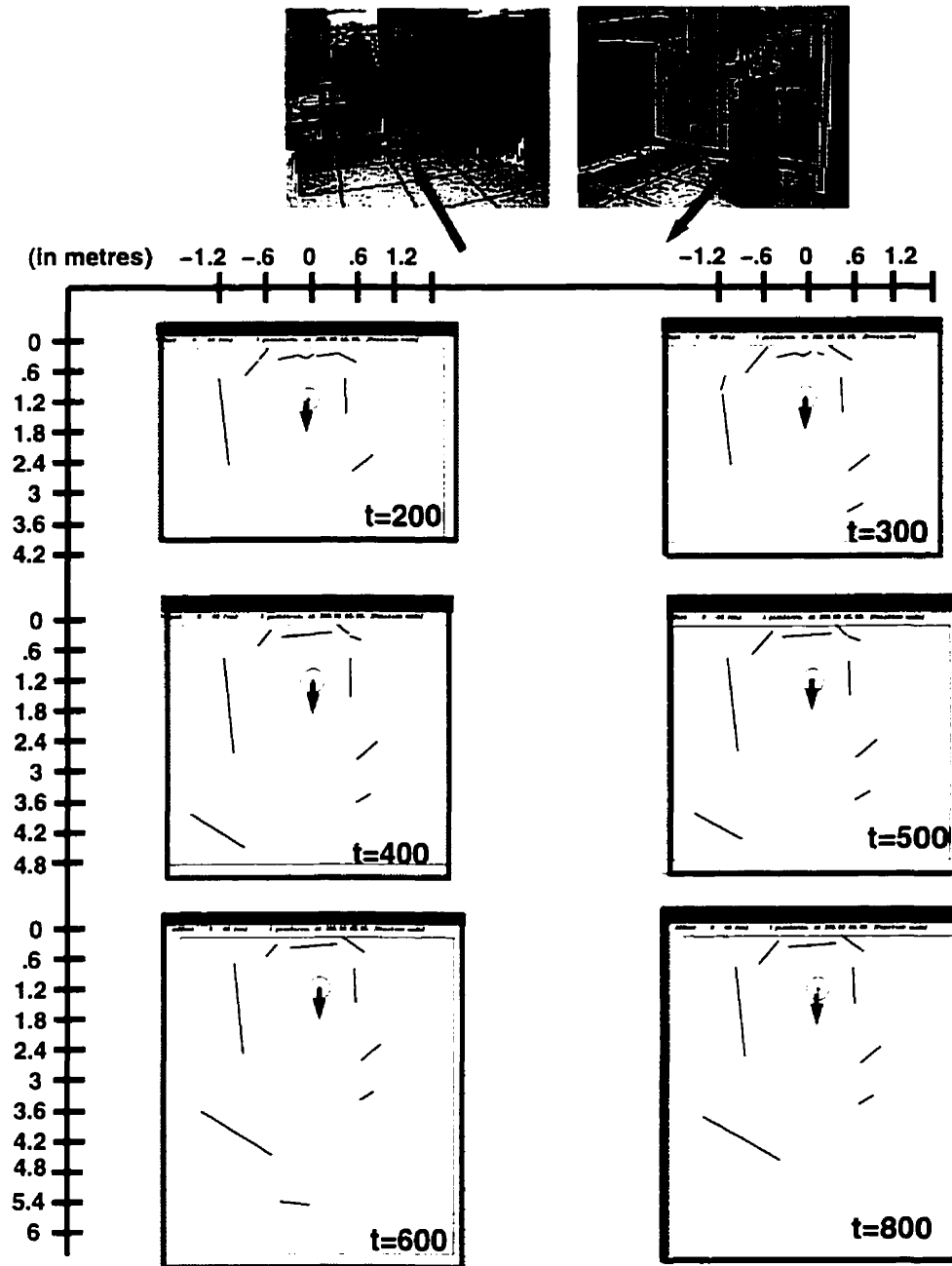


FIGURE 7.5. Sonar Mapping in Open Spaces. The sonar sensors on the Nomadics 200 have a range of 8 m. This figure illustrates mapping based on sonar data clustered into line segments (Mackenzie & Dudek, 1994) with different thresholds for discarding data points which fall beyond the specified range (i.e., threshold). The top left picture shows what the robot sees from its current position and orientation. The top right picture shows a picture of the robot with an arrow indicating its current orientation. The numbers in each map indicate a threshold value in cm: all sonar data points which fall beyond this threshold range are discarded as being unreliable. For an open space, such as the illustrated laboratory setting, the threshold value can approach the sensing range of the sonar sensors (e.g., 8 m) because there are very few unwanted multiple reflections.

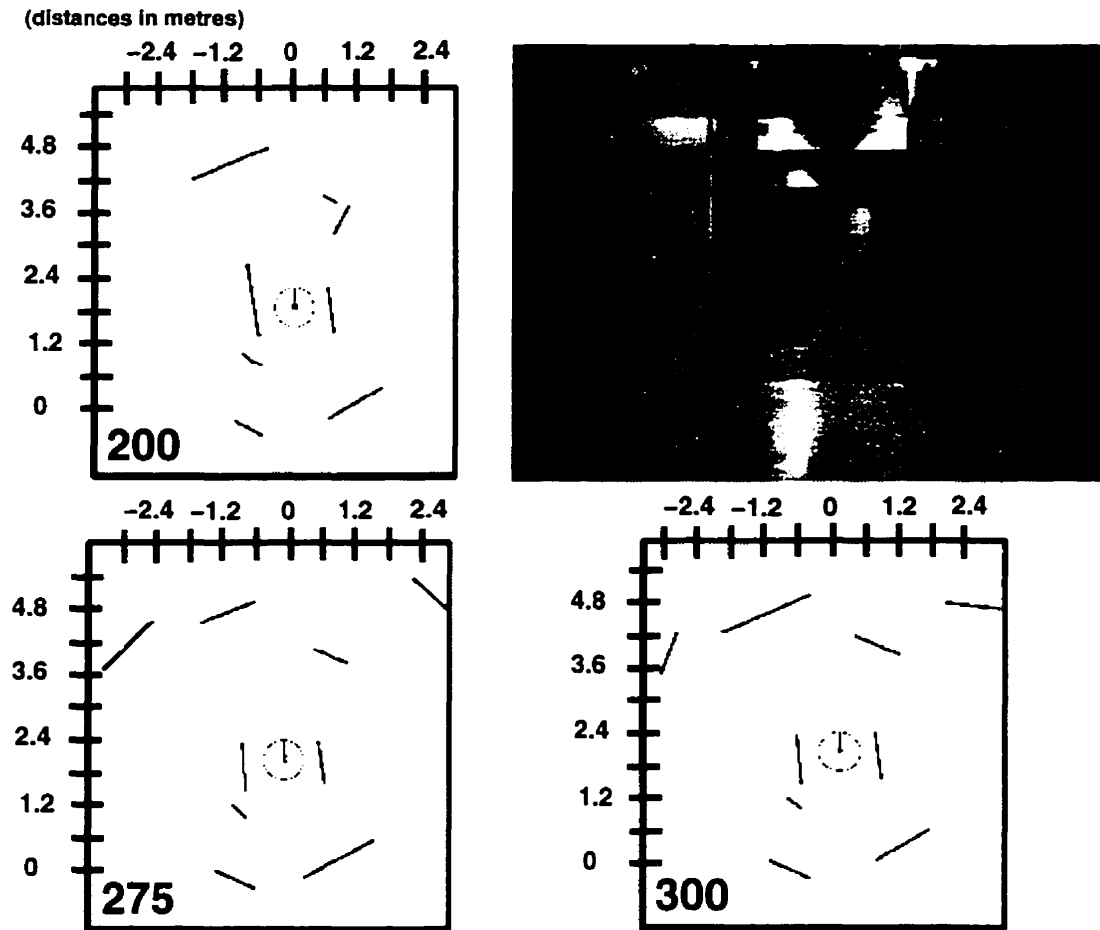
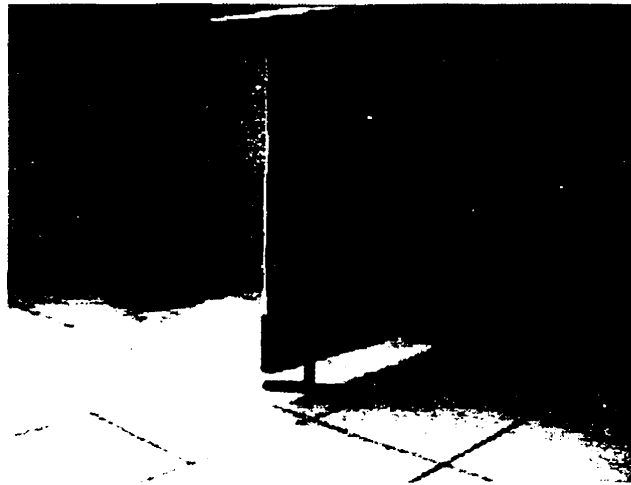


FIGURE 7.6. Sonar Mapping in Narrow Hallways. The sonar data is reflected many times in this environment making it difficult to use for map production. Reflections are due to the narrow corridor, door indentations, and corners. The number in each map indicates a threshold which dictates the maximum range (in cm) for which sonar data points are considered reliable in order to cluster them into line segments. Sonar data points beyond this threshold are not clustered.

### 3.2. Localization Using Sonar

The localization procedure takes a sonar scan and correlates the data with the CAD map to determine the pose (i.e., spatial location and orientation) of the robot (Mackenzie & Dudek, 1994). The only locations where this will work reasonably well are places where there are distinct features as seen by the sonar sensor. In an indoor environment, these locations are unobstructed corners (see Figure 7.7). Corners in offices are not good locations because there is usually a lot of clutter in offices and this will confuse the correlation process. An a priori CAD map (or a previously created map) is necessary as a reference to execute the localization procedure. Good localization regions are identified a priori in the CAD map. When the robot is within one of these regions, as the robot navigates about in an attempt to achieve its task objective, the robot is localized. It is assumed that before localization is performed, the estimated position of the robot is approximately equal to the correct position (i.e., within 1 m).



**FIGURE 7.7. Good Locations to Perform Localization.** Localization is performed in locations in the environment which are uncluttered and have distinct features (e.g., the corner next to the robot). The illustrated example shows the robot in an uncluttered corner. Sonar measurements are correlated with an a priori CAD map to determine the pose of the robot.

Potential problems with this technique include an aperture problem: certain regions in the environment appear the same to the sonar sensors (e.g., doors repeat, corners are similar looking). This is why the assumption of being near the true estimate is necessary. Another problem is that the actual corner is not properly mapped using the sonar data (see Section 3.1) and it is really only the two straight line segments defining the corner which are used in the correlation process. A problem may occur when the corrupted corner feature captured by the sonar data confuses the correlation process. A practical problem

with the sonar-based localization procedure (Mackenzie & Dudek, 1994) is its extremely long computation time. The process takes approximately half a minute and the robot is stopped during this procedure.

The robot localization error (i.e., if only dead reckoning is used) for a traversal of 3 m forward and 3 m backwards on a tiled floor (i.e., 1 m square tiles) has been found to be approximately 6 cm. Based on this error, and assuming that an error tolerance of 20 cm is the maximum that can be acceptable, the robot should be localized every 20 m of traversal<sup>15</sup>. It was found that the robot was localized within  $\pm 5$  cm of its actual position when the localization procedure based on correlating sonar data to the CAD map was executed. Future work should try to emphasize fast localization procedures which can function in real-time (i.e., concurrently done during the navigation task).

### 3.3. Mapping Using QUADRIS

QUADRIS (see Figure 7.8) is a system consisting of two range sensors (BIRIS) (Blais *et al.*, 1991) mounted on pan-tilt heads (Bui, in preparation). QUADRIS is mounted on the top of the basic Nomad platform. The BIRIS Range Sensor (Blais *et al.*, 1991) is a compact optical 3-D range sensor developed by the NRC (Canada's National Research Council) which is based on the use of a double aperture mask in place of the diaphragm of a standard camera lens. This laser line, when viewed through the double aperture, produces two lines whose separation correlates with the *range distance*.



FIGURE 7.8. The QUADRIS System. QUADRIS (see Figure 7.8) is a system of two range sensors (BIRIS) (Blais *et al.*, 1991) mounted on pan-tilt heads (Bui, in preparation).

QUADRIS' PTU's follow a scanning pattern (i.e., series of orientations of the cameras) which is based on the context of the current environment. Experiments have been conducted in an office and laboratory environment and two different scanning patterns were chosen; one when the robot is in a room and the other when it is in a hallway. In a room, the

<sup>15</sup>SPOTT's initialization parameters permit the frequency of localization to be based on time, the distance traversed, or a combination of the two. Currently, SPOTT will localize the robot after traversing at least 20 m and when the robot is in one of the localization regions.



QUADRIS sensor's direction is alternated between pointing forwards and sideways. In a hallway, the emphasis is more on pointing towards the sides. The QUADRIS data have been found to be very reliable<sup>16</sup> up to 2 m; however the measurements can be very sensitive to lighting conditions (Bui, in preparation). The reliability of the range data is dependent on having a credible lookup table (i.e., calibration) from which to correlate range and disparity values. An example of the type of disparity data and map produced from a flat wall segment is illustrated in Figure 7.9. Typical processing times are 900 msec for the camera selection and 400 msec for the disparity computation<sup>17</sup> and conversion into real world coordinates (Bui, in preparation).

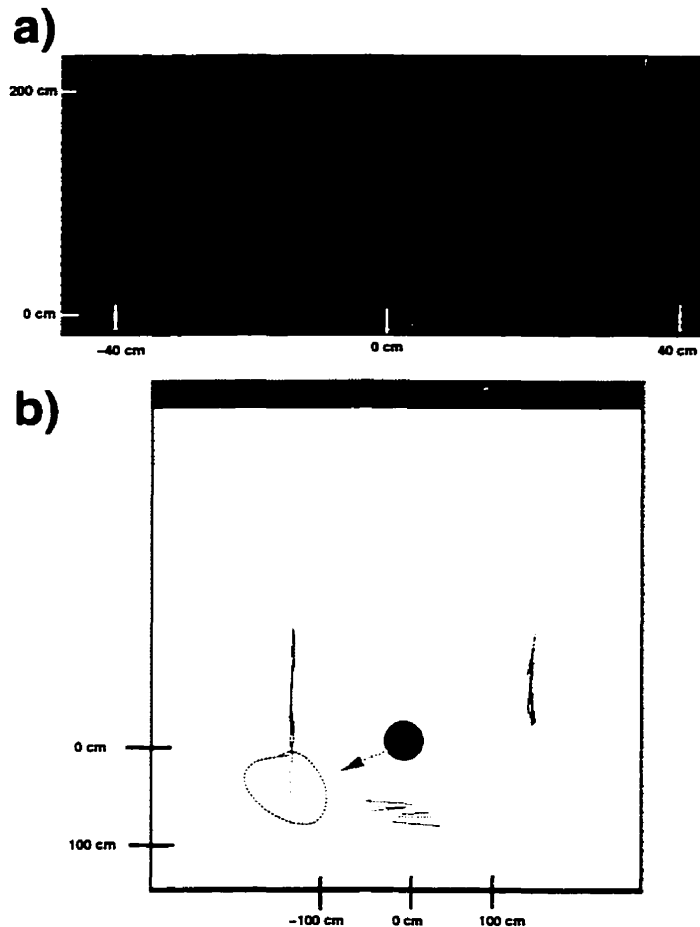


FIGURE 7.9. **QUADRIS Range Data.** During this experiment the robot was moving in a straight line in a rectangular room void of any obstacles. (a) shows the disparity returned for a particular scan. The scan direction is shown by the arrow on the robot in (b). (b) shows the map solely produced from QUADRIS data. The line segment circled corresponds to the disparity data in (a).

<sup>16</sup>QUADRIS can obtain range data for objects up to 5 m away but it is not always reliable.

<sup>17</sup>This also includes the overhead associated with monitoring the features to see if they correspond to objects QUADRIS can recognize (e.g., doors, walls).

The QUADRIS system appears to be give reasonably reliable range data at a rate of approximately 1 frame per second. It would be desirable to have the system return range data at a faster rate. The limiting factor appears to be the selection and control of the PTU units (Bui, in preparation), which is the attention problem<sup>18</sup>.

---

<sup>18</sup>The attention problem takes into account the speed limitations of the cameras mounted on the pan-tilt heads, in addition to the environment and task context to determine where to look (i.e., scan, point the cameras). This problem has been short-circuited by fixing the scanning patterns for when the robot is in a room and hallway. Currently, the camera selection is constrained to be no faster than 900 msec (Bui, in preparation).

#### 4. Navigational Experiments

The TR+ interpreter was found to take on average 0.7 msec/cycle for a tree with 20 conditions using between 10 to 15 processors. When 45 conditions are used, the average time was found to be approximately 1 msec/cycle. The time taken to converge to a solution by the local path planner, using a 75 by 75 grid on a collection of 5 SGI workstations for 3 obstacles and a goal, is approximately 4 seconds. For all experiments, the grid size for the potential field path planner was 35 by 35, which converges in less than 2 seconds.

Experimentation has been conducted for a variety of tasks. The experiments have been performed under two different scenarios: (1) no map is available a priori; and (2) a partial map of the permanent structures (i.e., walls) is available a priori. An *AUTOCAD* architectural drawing of the CIM office and laboratory space was used as a partial map of the permanent structures.

One experiment was performed using the TR+ program illustrated in Figure 3.12. There was no a priori map. The robot moved at 10 cm/sec and updated its map of the environment from sonar data every 5 seconds. The robot was stopped for approximately 5 seconds while acquiring and processing the sonar data. The robot was localized every 10 seconds<sup>19</sup>. The map built up during the execution of the task is given in Figure 7.10. Figure 7.11 shows selected frames during the execution of the task.

A second experiment, shown in Figure 7.12, illustrates the robot navigating to a pre-defined location. In this experiment, the robot uses an available CAD map as a priori knowledge. However, during task execution the robot discovers features (e.g., the partition) that are not present in the CAD map. In this example, the robot moved at 50 cm/sec. The mapping of the environment was performed every 150 cm traversed by the robot, or when previously unsensed parts of the environment had to be observed<sup>20</sup> (e.g., around the partition in Figure 7.12). Localization of the robot was performed at the start of task execution.

In addition to using sonar data for navigation, QUADRIS range data is also used, as well as infrared and bumper sensor data. QUADRIS range data is employed for simple object recognition. Horizontal and vertical object range profiles are used to recognize features such as doors, chairs and desks (Bui, in preparation). The control of the pan-tilt heads is a predefined scanning pattern depending on the context<sup>21</sup>. The bumper and infrared sensors

<sup>19</sup>This experiment was carried out when the localization frequency could only be specified with respect to time. Currently, the rate can also be specified with respect to distance traversed.

<sup>20</sup>This is done by observing the global changes in the robot's trajectory. If the robot's direction has changed by  $\alpha$  degrees since the last mapping, the environment is mapped again. The value used for  $\alpha$  was 60 degrees.

<sup>21</sup>For example, a different scanning pattern is used for hallways and rooms.

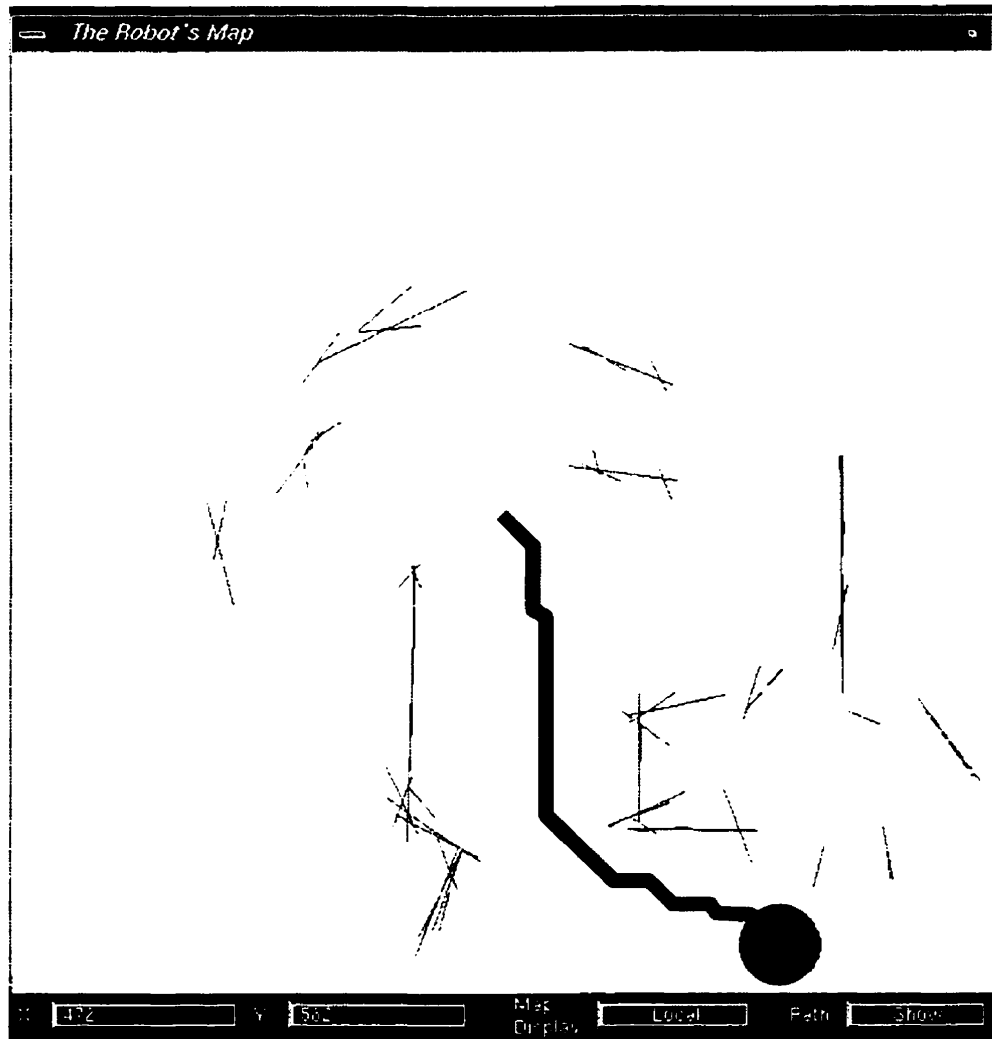


FIGURE 7.10. **Dynamic Mapping and Trajectory Determination.** As the robot moves towards a predefined spatial location, sonar data updates the map and alters the trajectory.

are used to indicate features missed by the sonar and range sensors, if any. The TR+ programs for QUADRIS (see Figure 3.16), bumper and infrared mapping (see Figure 3.18) are all similar to the “*Map\_the\_environment*” program illustrated in Figure 3.12. All of the mapping TR+ programs operate in parallel.

All of the TR+ programs discussed so far are based on knowing the spatial location of the goal beforehand. The task is completed successfully when the robot arrives at the desired position, as is the case when the task command is “*GO*”. However, when the task command is “*FIND*”, a search needs to be performed in the environment. The TR+ control structure can be used to guide the robot. Figures 3.14 to 3.22 in Chapter 3 show examples of the TR+ programs used for searching the environment to find a particular object. For the “*FIND*”



**FIGURE 7.11. Autonomous Navigation with No A Priori Map.** The robot successfully navigated around various obstacles as it traversed towards a predefined spatial location.

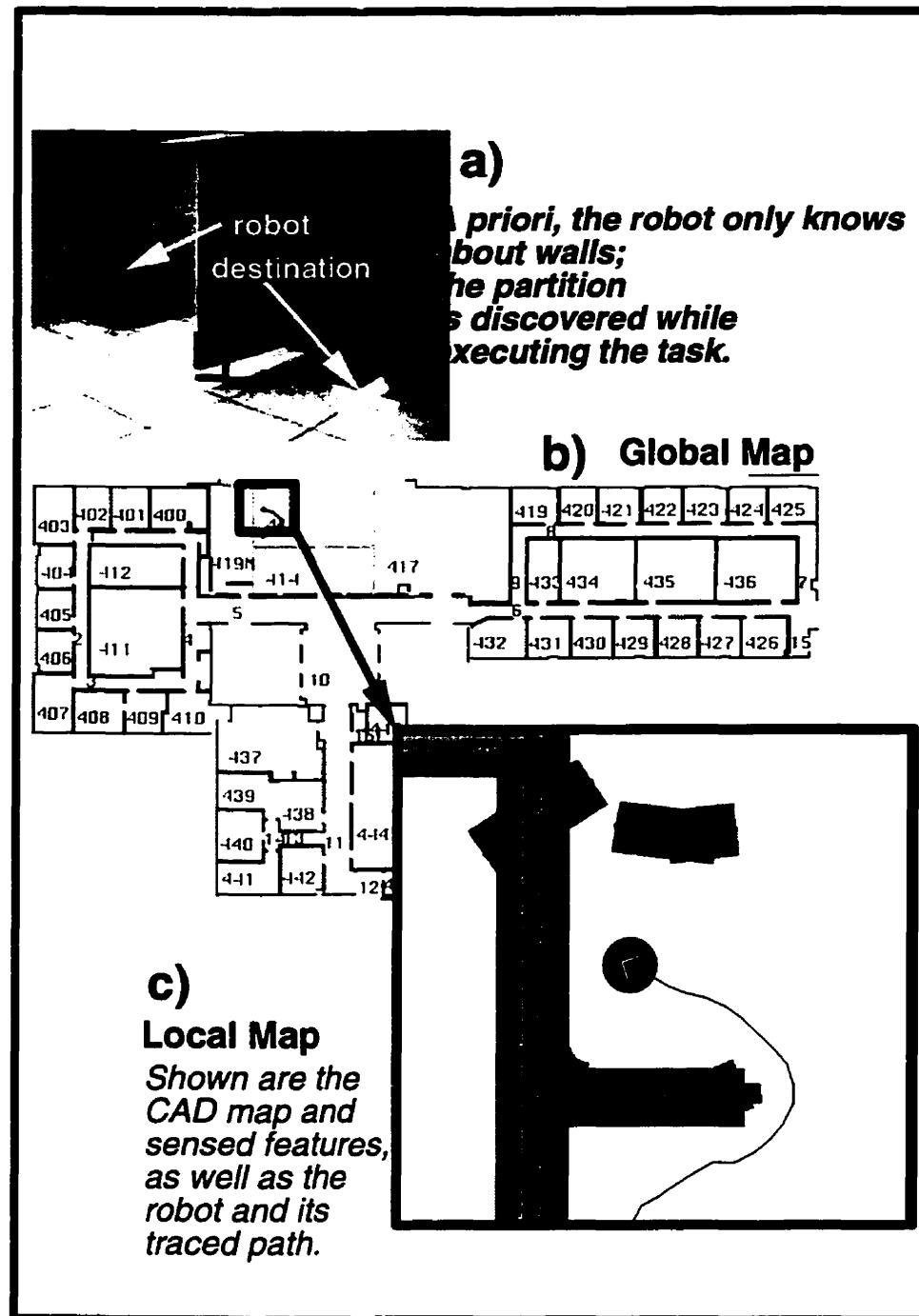


FIGURE 7.12. Autonomous Navigation with a Partial A Priori Map. A priori, the robot only knows of the walls in the CAD map. (a) shows the initial configuration and the desired destination. (b) shows the location of the local map in the global map. (c) shows the local map at task completion, along with the CAD (light lines) and sensed features (darker lines), as well as the executed trajectory. The padding of the features, to account for uncertainty and the local path planner modeling of the robot as a point, is illustrated by the grey regions surrounding the line (i.e., wall) features.

task, the types of objects which can be found is limited by the recognition capabilities of QUADRIS (i.e., range data). Recognition at this time is limited to chairs, doors and walls. Since the focus of attention problem has not yet been solved, the recognition of a chair also requires the robot to be in a particular position and pose with respect to the chair (Bui, in preparation). Object recognition based on QUADRIS data is an ongoing research project at CIM. The testing of the “*FIND*” task was only performed in simulation mode. One of the future roles for COCOLOG is to help SPOTT guarantee task completion for the “*FIND*” task (see Figure 1.3).

There are two potential problems with the path planner that were discovered during experimentation.

- The first problem is concerned with proximity sensor mapping (e.g., bumper, infrared). The features mapped by the proximity sensors are very close to the robot: so close (e.g., bumpers are contact sensors) that when these features are padded by the local path planner (i.e., to compensate for modelling the robot as a point), the current estimate of the robot’s position may actually fall on the plateau (i.e., saturated high value in the potential function) formed by the feature. There is no gradient on the plateau and therefore no trajectory is generated. To alleviate this problem, the steepest descent gradient is computed by using a window that takes into account the padding of the obstacles<sup>22</sup>.
- The second problem concerns the projection of the goal onto the local path planner’s border (see Figure 7.13). If an obstacle is discovered near the border of the projected location, it may have the effect of blocking the attractive forces of the goal. One potential solution is to project the goal onto the entire side of the potential field (e.g., top horizontal). When the global goal is projected as a point onto the local border, it is already based on the global geometry of the rooms to determine its precise placement. This has the effect of producing a smooth path when the local window is moved. When the global goal is projected onto the entire side, the global path will not necessarily be as smooth<sup>23</sup>. Also, there is still no guarantee that newly discovered obstacles will not block the attraction generated by the entire side. It may be that an additional role for the reasoning agent (i.e., COCOLOG, see Section 3 in Chapter 6) is to monitor if the current projected goal is blocked due to

<sup>22</sup>The size of the window used to compute the gradient is usually 3 by 3. A plateau is identified if there is no gradient at the current robot position. If such is the case, a 5 by 5 operator is used to compute the gradient. The size of the alternative operator is based on the padding size (i.e., 25 cm, 50 cm for both sides of a line), and the pixel (i.e., grid element) size (i.e., 15 cm).

<sup>23</sup>The smoothness of the global path will depend on where the path exits each local window.

the dynamically sensed data and to determine a new projected goal (see Section 9.3 in Chapter 4) if this happens.

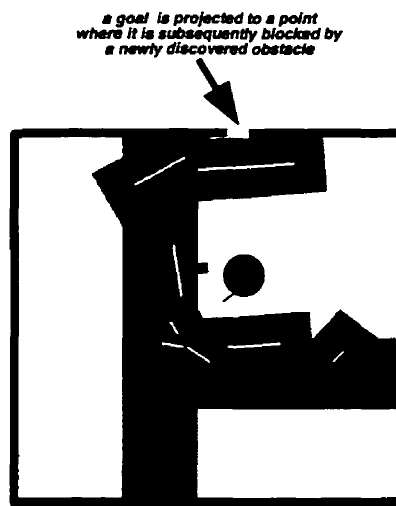


FIGURE 7.13. **Projection Onto a Point or a Line.** The global goal is either projected onto a point on the border or onto the entire side of the potential function (e.g., top horizontal). The projection onto a point provides a better estimate of where to direct the robot towards the global goal. However, if an obstacle is discovered at the border of the potential function, there is a potential for the attractive forces of the goal to be blocked. In this diagram, the light lines are newly sensed features, while the dark lines were obtained from the CAD map. The location of the illustrated projected goal is such that it has no attractive pull on the robot when the global goal is projected as a point.

A troublesome engineering problem experienced while navigating in the office environment was the loss of communication with the robot via the radio ethernet wireless link. There is also a video link to transmit the image data for obtaining range from the QUADRIS system<sup>24</sup>, whose communication was also degraded at certain times. The degradation usually occurred when there was no line of sight from the transmitter on the robot to the receiver and the distance exceeded 40 m. Another engineering issue is the battery life of the robot, which is about three to four hours. This can be a problem for continuous operation when the robot is required to be active for longer periods than the life of the batteries (e.g., night surveillance of an office building).

When existing and self-contained modules (e.g., sonar mapping) are integrated into a larger system (e.g., control system such as SPOTT), they usually require some redesign. The sonar mapping and localization packages were never intended to operate in real-time, and should be modified so that they are able to do so.

<sup>24</sup>Ideally, the QUADRIS data should be processed on board the robot. The image consists of two projected laser lines, whose separation correlates with the range distance. Only the range data values are of interest. The rest of the video data are discarded. If this computation were performed on board, then the range data could also be sent via the radio ethernet link.



## 5. Off Site Experiment

The entire SPOTT system was demonstrated at the 1996 IRIS-PRECARN conference, held at the Queen Elizabeth Hotel in Montreal, Quebec, Canada between the dates of June 4-th through to the 6-th, 1996. The robot as well as three workstations were taken to the hotel. An optical fibre ethernet link connected the workstations to eight other workstations at the CIM laboratories at McGill University. The demonstration was to show SPOTT's navigational capabilities in an office environment. COCOLOG was also used as an interface to the SPOTT system. However, it did not contribute anything new as it only took the place of SPOTT's existing global path planner. The interesting aspect of COCOLOG's involvement was that it was integrated with SPOTT. COCOLOG generates a path by setting intermediate goals at edges (i.e., doorways). SPOTT completes each intermediate goal, notifies COCOLOG that the goal has been reached, and waits for COCOLOG to return another intermediate goal.

The office environment was produced using interlocking partitions, and a map (see Figure 7.14) was available to SPOTT. The entire demonstration also served as an experimental testing ground for the SPOTT system and its components (see Figure 7.15).

The main problem experienced during the demonstration was with respect to the robustness and reliability of the sensor data, namely sonar and QUADRIS. The partitions were connected with tubes that had many ridges notched into them. These notches produced multiple reflections with the sonar data (see Figure 7.16) and had the effect of artificially blocking the doorways in the potential field. This problem was temporarily addressed by reducing the padding factor to make sure that the door was not blocked in the potential field by the sonar data returned from the notched tubes. Future work with sonar data processing should concentrate on modelling and compensation for various materials and types of surfaces and their geometric configurations. Reliability can be maintained by possibly using the CAD map to verify if the returned sonar data is an accurate reflection of the environment. QUADRIS also experienced some problems by producing features that were the result of lighting effects as opposed to actual physical objects. The lights in the room had to be dimmed and the curtain was drawn. Future research with QUADRIS should concentrate on its robustness in different lighting conditions.

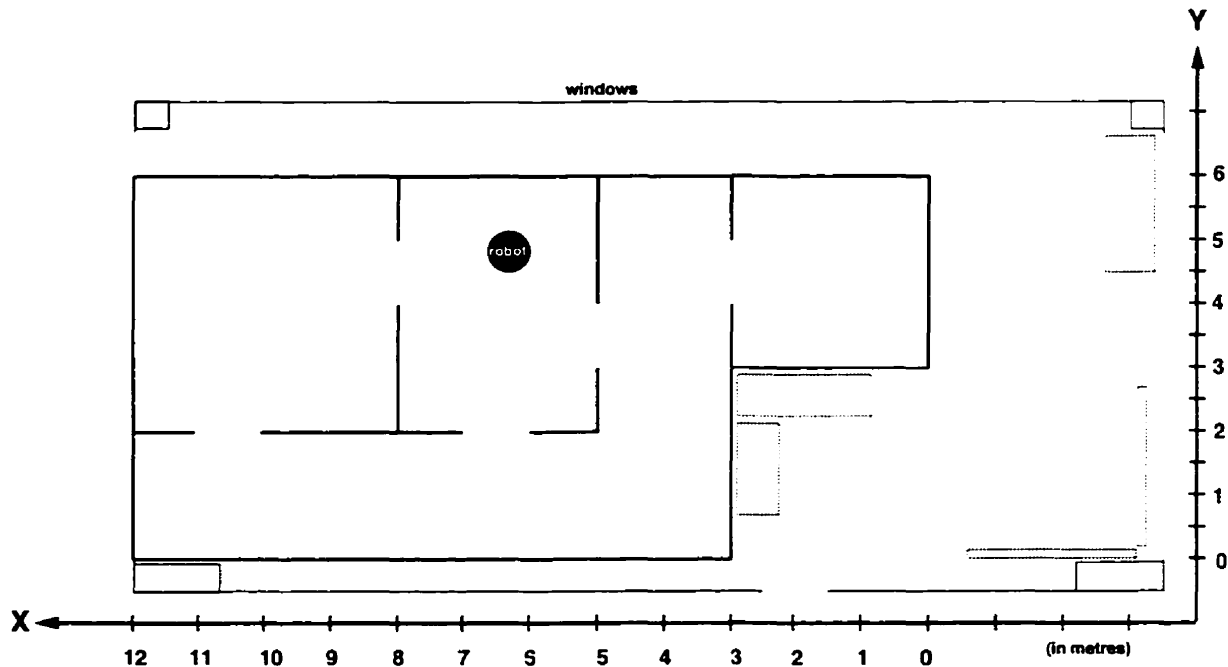


FIGURE 7.14. Map of the Demonstration Environment. This is the a priori map given to SPOTT for the demonstration at the 1996 IRIS-PRECARN conference held in Montreal (June 1996).

Most of the difficulties with making the demonstration work concerned engineering problems, and have been or will be corrected with simple software changes or bug fixes<sup>25</sup>. They are as follows:

- The *robodaemon* process (i.e., provides low level control of the robot) blocks communication with the robot until the sonar data collection is complete. In addition, when sonar data collection was initiated while the robot was moving, *robodaemon* lost track of the robot's position.
- There were occasional problems in allocating sockets for communication with the two daemon processes responsible for controlling the QUADRIS PTU's and acquiring QUADRIS range data.
- The global graph planner (i.e., when not executing COCOLOG with SPOTT) occasionally returned an incorrect path. This was traced to a problem with how communication is done with PVM. PVM guarantees processing the messages in the order in which they arrive at their destination and not necessarily in the order they

<sup>25</sup>There were a series of problems that were encountered leading up to the demonstration. The week before the demonstration, three circuit boards on board the robot failed. The boards were replaced a couple of days before the demonstration. This prevented extensive testing of SPOTT and COCOLOG before the conference. In addition, the conference organizers failed to provide a couple of sample partitions so that testing could be done before the demonstration. There was also only 2 hours available once all the equipment was moved to and set up at the hotel for testing the system.

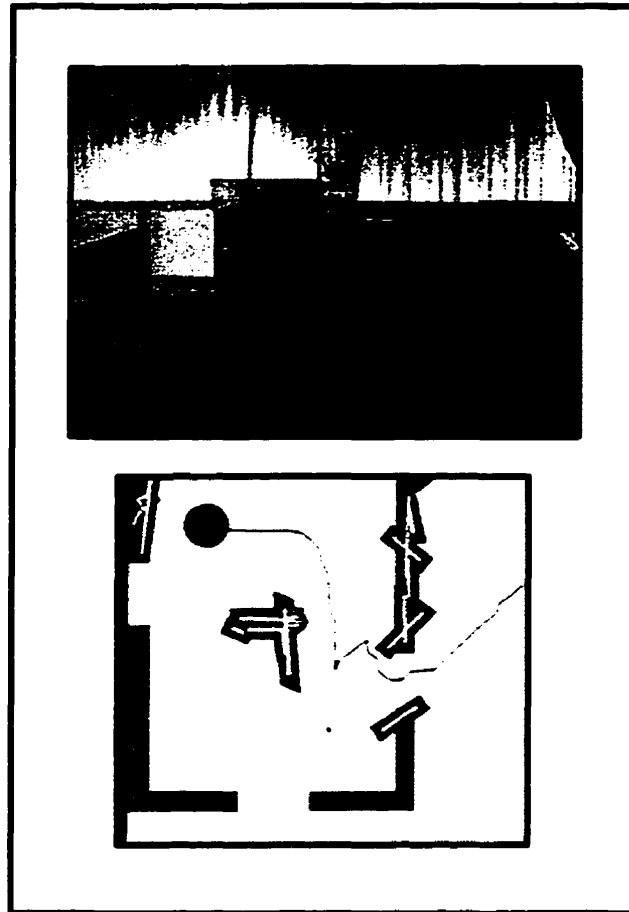


FIGURE 7.15. Example of an Execution During the IRIS-PRECARN Demonstration. SPOTT is aware of the walls a priori. COCOLOG is responsible for setting intermediate goals (i.e., doorway accesses). SPOTT's role is to get the robot from one doorway to another. The top diagram illustrates the robot navigating around the box and the bottom diagram illustrates the map produced from the newly sensed data (i.e., bright lines) and the existing CAD features (i.e., dark lines). All of the map features (i.e., lines) are padded with a grey border to show how the path planner compensates for modelling the robot as a point.

are sent. A message was sent to update the global graph and subsequently a request (i.e., from the same sending process) was sent requesting the current global path. The update and the request were sent from the *GUI* process. Sometimes the request for the global path was processed before the update had been processed. Integrating the update and path request into a single message packet will alleviate this problem.

- There is a real need for a comprehensive set of debugging resources. These resources can either be temporary data files (i.e., verbose descriptions of what the process is doing) or graphical visualizations of execution. Some of these resources are already in place and were a very useful resource when debugging specific problems.

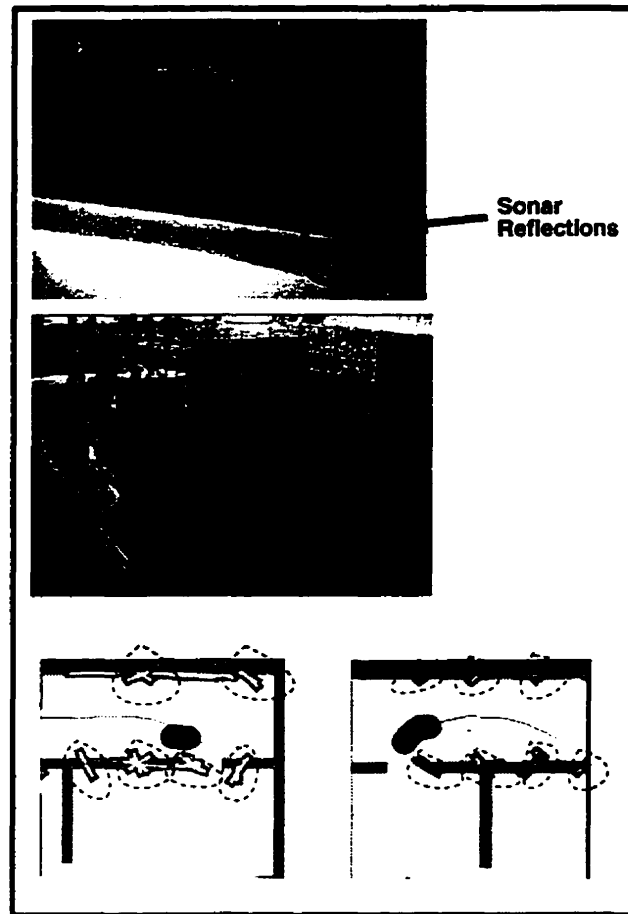


FIGURE 7.16. **Sonar Reflections.** The poles which were used to keep the partitions in place had little ridges notched into them which caused the sonar data to be reflected many times and return confusing and erroneous map data. The top two diagrams illustrate the robot in its environment. The bottom two diagrams (i.e., side by side) illustrate the dynamically produced map during the navigation task. The circled features show how the sonar data was reflected at the ridged poles. The produced map should have contained only horizontal and vertical lines.

- COCOLOG requires a lot of computation time to compute such simple things as path planning. This is because of the overhead associated with its logical reasoning and theorem proving. SPOTT's global path planner is able to generate a path of 10 nodes to be visited (i.e., based on the graph in Figure 4.13) in less than 200 msec, while at present COCOLOG requires over a minute in initialization time, and a subsequent 5 to 10 sec for computing each subsequent intermediate goal. In order to make COCOLOG viable in a real-time setting, practical operational issues have to be taken into account. COCOLOG should be able to precompute some of its results (i.e., when possible) and store them in memory. This is in contrast to computing the result only at the time when it is requested (i.e., by SPOTT) to do so.

The experiments have demonstrated that SPOTT is able to perform navigation in an unpredictable environment which may be partially known a priori. There are some research issues that require attention, namely the ability to provide reliable and robust map data from the sonar and QUADRIS sensors. Most of the naive visitors to the demonstration also queried why the robot moved so slowly<sup>26</sup>. The execution of the robot is only as fast as its slowest component and the slow elements appear to be the processes responsible for processing the sensor data and the low level robot controller (i.e., *robodaemon*).

---

<sup>26</sup>The robot moved at the rate of 15 cm per second, but it stopped to collect sonar data every 1 m traversed. The robot was stopped for approximately 5 seconds during sonar data acquisition and processing.

## 6. Lessons Learned

The experiments performed with the SPOTT system helped identify the following issues:

- SPOTT relies heavily on its message passing implementation. The speed of processing messages is dependent on the size of the message and the speed of the lines of communication. Communication speeds can improve drastically if other protocols such as Fast Ethernet or ATM are used instead of Ethernet. Other ways of communicating between processes that should be investigated include shared memory, and lightweight processes (i.e., low overhead) called *threads*. There is a thread version of PVM called TPVM which is a current research project (Ferrari & Sunderam, 1995).
- Mapping the environment with sonar and QUADRIS data currently occurs independent of each other and other available information. Sonar data is prone to be bad in areas where multiple reflections occur (e.g., corners). The a priori CAD map can be used to provide some verification of the sonar data. Similarly, the QUADRIS system needs to be made more robust to varying lighting conditions. The experiments showed that these sensors are prone to give potential erroneous results in different environments. Sensor algorithm testing should always try to vary some elements of the environment (e.g., wall types, lighting).
- There is a real need for real-time localization: fast algorithms for correcting the robot's position at regular intervals (i.e., every 20 m of traversal).
- Dynamic mapping may cause the attraction of a global goal to be blocked within the confines of the local potential field. It is not clear how to guarantee that this does not occur with a simple algorithm, without requiring a reasoning agent to monitor and compensate for it.

There is never enough experimentation one can do with a mobile robot control system (e.g., SPOTT). All the unknowns in the environment can never be fully anticipated. The SPOTT system has been proven to be able to navigate a robot in an office and laboratory environment. The current issue is how to make the robot move faster: an ideal speed for the robot would be the average walking pace of a human (i.e., approximately 1 m/sec). Related to the issues of speeding up the robot's motion and the acquisition of sensor data (i.e., sonar, QUADRIS), there is an engineering issue to address (i.e., related to the speed): the low level robot control software should be open to multiple commands and requests from

various processes and concurrent operation (e.g., moving and acquiring sonar data). At normal walking speeds, the speed of the local path planner<sup>27</sup> will also have to be reassessed.

---

<sup>27</sup>Recall that the local path planner takes two seconds to converge to the harmonic function for a new configuration.

## CHAPTER 8

---

### Conclusions

This thesis has presented an architecture - called SPOTT - which provides a bridge between behavioral and symbolic (i.e., reasoning) control. Figure 6.8 illustrates one way of potentially integrating a reasoning system (e.g., COCOLOG<sup>1</sup> (Caines & Wang, 1995)) with SPOTT. In addition, SPOTT is modular and programmable, which permits the possibility of extending the current functionality, as well as the domains of execution. New control strategies as well as additional sensors and actuators can be easily incorporated within SPOTT. There is a wide variety of navigational tasks which SPOTT can execute based on its defined task command lexicon (see Chapter 5). With a planning module, SPOTT is able to guarantee navigational task completion under many different scenarios (see Figure 1.3). This is what primarily distinguishes SPOTT from other behavioral-based robot control architectures (e.g., subsumption architecture (Brooks, 1986)). SPOTT consists of a behavioral controller, a local path planner, and a global path planner. The behavioral controller and the local path planner are both in a real-time<sup>2</sup> feedback loop with the robot and its environment. The global path planner is responsive to slower changes in the environment (e.g., robot changes its position with respect to a room).

Behavioral control is based on extending an existing formalism - Teleo-Reactive programs - into Teleo-Reactive+ (TR+) programs (see Chapter 3). SPOTT is the first implementation which experiments with a real-time version of the TR formalism (i.e., called TR+). The latter is not limited to reactive behaviors (e.g., *bumper collision detection*,

---

<sup>1</sup>The logical reasoning system COCOLOG (Caines & Wang, 1995) has actually been interfaced with the SPOTT system (see Section 3 in Chapter 6). COCOLOG did not contribute anything new to SPOTT as it only took the place of SPOTT's global path planner. The interesting aspect of COCOLOG's involvement was that it was integrated with SPOTT. Future uses of COCOLOG should address the elements of navigation for which SPOTT cannot guarantee task completion, as suggested by Figure 1.3, as well as providing capabilities such as determining goal reachability, spatial reasoning and map maintenance, and the learning of new TR+ programs (i.e., control strategies).

<sup>2</sup>SPOTT reacts faster than the times taken by changes in the environment.



*failure monitoring*), but also permits functional behaviors (e.g., *mapping the environment, localizing the robot*), as well as symbolic-like behaviors (e.g., *are all the rooms explored*).

A potential field approach is used for dynamic<sup>3</sup> path planning. The potential field is used as a local path planner because its computational requirements increase proportionally with the number of grid elements, and because long narrow corridors (i.e., prevalent in indoor environments) do not permit the rapidly decaying potential function to be fully representable in a 32-bit computer address (see Section 7 in Chapter 4). The harmonic function is computed using an iterative approach (see Section 4 in Chapter 4). The robot's trajectory is determined by performing steepest gradient descent on the computed harmonic function (see Section 5 in Chapter 4). An algorithm is proposed which will guarantee proper control given that real-time execution and computation are to be done concurrently (see Section 6 in Chapter 4). In order to guarantee proper control, a collection of potential fields at varying grid resolutions are computed.

If a graph abstraction of the environment (i.e., CAD map) is available (i.e., rooms are nodes, access ways are arcs) a priori, an algorithm is presented which projects the global goal (i.e., outside the extent of the local potential field bounds) onto the potential field boundary (see Section 9 in Chapter 4). The algorithm makes use of the global path (see Section 8 in Chapter 4) produced by the global path planner (i.e., using the abstract graph map). When the robot is close to the boundary of the potential field, the potential field is moved so that it is centered around the robot's current location.

A real-time and parallel implementation of a TR+ interpreter and a dynamic real-time path planner have been developed (see Sections 2.3 and 2.5 in Chapter 6) using a message passing software package called PVM (see Section 1 in Chapter 6). Navigational experiments have consisted of moving the robot to known and unknown spatial locations with no or a partial a priori map (see Section 4 in Chapter 7). A priori maps are readily available for most indoor environments in the form of architectural CAD drawings. SPOTT's success to date gives optimism for experimenting with more complex environments as well as in other unstructured environments which may be hazardous or remotely located.

---

<sup>3</sup>Dynamic means that obstacles are discovered while executing a path. Hence, path planning and path execution are done concurrently.

## 1. Contributions

This thesis addresses the problem of autonomous navigation of a mobile robot in an indoor environment, such as an office or laboratory space. The navigational tasks are based on a language lexicon (see Chapter 5) within two contextual settings: (1) a partial map of the environment is available a priori; and (2) no map is available a priori. A robot control architecture called SPOTT is proposed and implemented as a real-time and parallel system of concurrently executing and co-operating modules. Inherently, the control system is a real-time Artificial Intelligence (AI) system which is responsible for dynamically adapting to changing environmental circumstances in order to successfully execute and complete a set of navigational tasks for an autonomous mobile robot. SPOTT consists of a behavioral controller, a local dynamic path planner, and a global path planner, as well as a map database and a graphical user interface. The SPOTT architecture provides a framework for the inclusion of additional sensors and associated perceptual processing algorithms, actuators and control strategies.

SPOTT is a novel robot control architecture because it proposes a way of linking behavioral (i.e., reactive) and symbolic control. SPOTT differs from other behavioral architectures by being able to guarantee task completion<sup>4</sup> under many different scenarios (see Figure 1.3). The exploitation of existing computational resources by the distributed implementation of SPOTT is additionally innovative. Contributions are also made in the following three areas:

- (i) The Teleo-reactive (TR) control (Nilsson, 1992) formalism forms the centerpiece for the behavioral controller (see Chapter 3). This thesis has contributed to TR behavioral control research in the following manner:
  - This thesis is the first research work to design, implement and test an on-line and distributed TR interpreter - of concurrently executing behaviors - to handle real-time control of an autonomous robot.
  - The basic TR language is extended to handle multiple goals, concurrent actions, and conditional expressions. The extended formalism is called TR+.
  - A typical TR+ program for navigation is presented in Figure 3.13. This provides a guideline for writing TR+ navigational programs when executed in the SPOTT architecture.
- (ii) Path planning (see Chapter 4) is concurrently performed at two levels of abstraction. The local path planner is a potential field approach based on a harmonic function.

---

<sup>4</sup>Most behavioral architectures are not able to predict successful task completion for any situation.

It is guaranteed to find a path to a goal if such a path exists. The local path planner is in the critical real-time control feedback loop with the environment. The global planner plans a path based on a graph abstraction of the environment. The contributions made to the field of path planning, in particular, the potential field approach using harmonic functions, are as follows:

- In the local path planner, path computation and execution are done concurrently. In order to guarantee a correct control strategy during concurrent plan computation, a hierarchical coarse-to-fine procedure based on a set of harmonic functions at varying resolutions is proposed (See Section 6 in Chapter 4).
  - A method is proposed for addressing the issue of planning for global goals when the extent of the potential function is limited (see Section 9 in Chapter 4). This is necessary because (1) the potential function is a rapidly decaying function which is not computable for all grid elements in a large array, and (2) the computation time increases proportionally with respect to the number of grid elements. Thus, the local path planner needs to receive *global goal* information from the global path planner. The global goal is projected onto the border of the local potential field. The boundaries of the potential field are moved when the robot approaches the current local extent, so as to centre the new bounds around the robot.
  - A method for computing the harmonic function in real-time and in parallel with existing computational resources is proposed, implemented and tested (see Section 2.3 in Chapter 6).
- (iii) This thesis is the first to propose and put to use a mobile robot task command lexical subset - consisting of verbs and spatial prepositions - (see Chapter 5), that is a minimal spanning basis set for human 2D navigational tasks (Landau & Jackendoff, 1993; Miller & Johnson-Laird, 1976). A procedure for quantifying the task command for execution in the behavioral controller and dynamic local path planner is presented. The quantification of the spatial prepositions is shown to depend on two norms. The two norms are the definitions for the spatial prepositions *near* and *far* in the current context of the environment and task.

## 2. Robot Control Issues

The design of a behavioral architecture for mobile robot control has to take into consideration the following issues (see Section 3 in Chapter 2):

- (i) How well does the architecture **scale** to more complex problems?
- (ii) Is the architecture **general** enough for a wide variety of tasks and environments?
- (iii) Are the executed actions of the robot **predictable** beforehand?
- (iv) Is the operator able to **monitor** the robot's interaction with the environment?

SPOTT has addressed these design issues in the following manner:

- SPOTT is *scalable* in the sense that it allows for new perceptual capabilities, actuator mechanisms, and behavioral decision rules without requiring the redesign of the underlying architecture. The system also suggests a way of interfacing to a reasoning module, and shows how this can be done by interfacing to an AI planner<sup>5</sup>. The COCOLOG (Caines & Wang, 1995) logical reasoning system has also been integrated with SPOTT proving that actual integration with a reasoning agent is possible (see Section 3 in Chapter 6). A TR+ control program concurrently executes many sensor-action control loops of varying execution time<sup>6</sup>. The behavioral control program language (i.e., TR+) is an extension of a language which is already being experimented with for learning control strategies (Nilsson, 1994), (Nilsson, 1995).
- SPOTT is *modular*. It consists of decomposable and replaceable parts. The main components are a behavioral control language interpreter, a dynamic path planner, and an AI-based planner. The dynamic path planner is local, and the AI-based planner is global. The dynamic path planner is only aware of a local window into the map of the environment at any particular time.
- SPOTT can be used for a wide variety of tasks and environments (i.e., *general*). The language of programming behaviors (i.e., TR+, see Chapter 3) and the software associated with the architecture remains the same from task to task. The control programs can also be easily modified. A repertoire of tasks for navigation has been defined which may be expanded in the future to incorporate manipulation tasks. In order to execute in real-time, SPOTT has been implemented using a message passing software package called PVM which transparently distributes and manages the processing across a collection of heterogeneous processors (see Chapter 6). The

<sup>5</sup>This refers to one of the many search algorithms which are available in the Artificial Intelligence literature. Specifically, Dijkstra's algorithm is used.

<sup>6</sup>This assumes that the time dependence is related to the amount of processing and modelling of the raw sensor data.

portability and heterogeneous property of PVM allows for the possibility of porting this architecture on board the robot in the future.

- SPOTT's actions are *predictable* at a certain level of abstraction. This is when the spatial location of the goal specified by the task is known (i.e., for the *GO* task). There is a level of uncertainty in the modelling of the environment, especially when little a priori information is provided (i.e., map). The exact path cannot be predicted, but the dynamic path planner can guarantee that the robot will circumvent an obstacle if it is actually possible to do so. The system also guarantees task completion in most circumstances which do not require extensive spatial reasoning or higher level cognitive skills. See Figure 1.3 for a list of tasks that SPOTT can guarantee successful completion and for which ones it requires assistance. Support can probably be obtained from an external reasoning agent (e.g., COCOLOG) or a human operator.
- A set of tools have been developed (see Chapter 6) for SPOTT so that the *monitoring* of execution and the programming of the behavioral control programs is made easier to *understand*. A graphical visualization tool has made the art of programming behavioral control programs relatively easy (see Section 2.5 in Chapter 6). The execution of the behavioral rules is visualized dynamically by a graphical representation of the rule base. The current state and path to be followed in the abstract graph map is also visualized dynamically (see Section 2.1 in Chapter 6).

SPOTT has addressed the problem of mobile robot navigation, in addition to paying attention to some of the general issues associated with the design of a robot control architecture. What distinguishes SPOTT from other behavioral architectures is the inclusion of a path planner into the real-time feedback loop with the robot and its environment. This permits SPOTT to predict task completion in certain situations. The SPOTT architecture is also able to scale to different problem scenarios (e.g., partial a priori map or no a priori map). New control strategies (i.e., TR+ programs) can be easily added without any modifications to the underlying architecture.

### 3. Future Research Possibilities

There are several research issues which should be explored in the future, namely:

- (i) For the task of navigation, writing a TR+ program is relatively simple because most of the navigational behaviors are independent of each other and may be executed concurrently (see Section 2.2 in Chapter 3). This would not be the case if the manipulation of objects were part of SPOTT's task repertoire. In this situation, there would be a strong dependence between manipulation and navigation (e.g., the robot has to be near the object before it can manipulate it). A guideline for writing a typical TR+ navigation program using the SPOTT architecture is shown in Figure 3.13. Future experimentation and research with manipulation tasks may provide a similar guideline for manipulation tasks. The use of TR+ programs for providing event driven control in other fields such as telecommunications, industrial process control, or manufacturing will require the definition of the necessary behaviors as well as a determination of their inter-dependence.
- (ii) It is theoretically possible to extend the path planning formalism presented in Chapter 4 for 2D path planning (i.e., local path planning) to address non-holonomic control, moving obstacles, and 3D path planning (see Section 10 in Chapter 4). One way to address extension into these three areas is to include another dimension into the potential field grid space to represent: (1) constraints for non-holonomic control (Connolly & Grupen, 1994); (2) time in order to capture obstacle dynamics for control with moving obstacles; or (3) the vertical dimension for 3D path planning. The addition of another dimension, while theoretically possible, is not practical in most cases<sup>7</sup>. The computational time for a harmonic function on a discretized grid will increase in direct proportion to the number of grid points. The extension of the presented techniques for path planning to these new areas is a research challenge that requires a practical solution given current technology.
- (iii) The task command lexicon is only defined for navigational tasks. If manipulation tasks were to become part of SPOTT's repertoire, then the task command lexicon would need to be expanded.
- (iv) It is desirable to make the robot move faster: an ideal speed for the robot would be the average walking pace of a human (i.e., approximately 1 m/sec). Currently, the robot moves at about 15 cm/sec, but stops for about 5 sec, every 75 cm of traversal in

---

<sup>7</sup>It is practical only when the discretized grid is coarse in 2D and will be coarse in the third dimension also (i.e., there are not that many grid points).

order to collect sonar data. The current hindrances to this are the speed of processing the sensor data and the limitation imposed by the low level robot control software in not being able to process concurrent actions (e.g., robot movement and sonar data acquisition) (see Chapter 7). The limitations of the low level robot control software is an engineering issue which is currently being addressed.

- (v) SPOTT relies heavily on its message passing implementation (i.e., PVM). Communication speeds can improve drastically if other protocols such as Fast Ethernet or ATM are used instead of Ethernet. Other ways of communicating among processes that should be investigated include shared memory and lightweight processes (i.e., low overhead) called *threads* (see Chapter 7).
- (vi) Harmonic functions (i.e., for the potential field, local path planning) can be computed in hardware using a resistive grid. This avenue of research has been pursued at the University of Massachusetts at Amherst and should be further investigated (see Section 4.1.1 in Chapter 4).
- (vii) Future modifications to SPOTT should address the issue of continual maintenance (e.g., fusing map features, abstracting new nodes from the sensor data) for the map database. This will require some reasoning capabilities which may either be part of the map database as a blackboard structure, or determined by an external reasoning agent (e.g., COCOLOG (Caines & Wang, 1995)) (see Section 2.4 in Chapter 4).
- (viii) SPOTT has been interfaced with the logical reasoning system called COCOLOG. Other future uses of the COCOLOG reasoning system (see Section 3 in Chapter 6 could include dynamically specifying changes to the TR+ programs during execution, possibly learning new TR+ control programs, learning the abstract graph map in environments where this is not specified beforehand<sup>8</sup>, and map database maintenance (i.e., verifying, refuting, and merging symbolic data constructs).
- (ix) One outstanding research issue which requires further attention is how to lay out the graphical information on the computer screen and present it to the operator. Figure 6.3 illustrates one such arrangement. However, there is not enough room on a single display to also show the global path and it is only displayed when the operator requests it. Additional graphics for visualization are usually displayed on another workstation (e.g., PVaniM for visualizing PVM memory and processor

---

<sup>8</sup>In the current implementation of SPOTT, the user creates the abstract graph map based on an architectural CAD map. It would be ideal to automate this process before execution, or to learn this abstract graph while building a map.

resource usage). The presentation of the graphics information has only been superficially addressed by this thesis and requires further examination (see Section 2.1 in Chapter 6).

- (x) Mapping the environment with sonar and QUADRIS data currently occurs independent of each other and other available information. Sonar data is prone to be bad in areas where multiple reflections occur (e.g., corners). The a priori CAD map can possibly be used to provide some verification of the sonar data. Similarly, the QUADRIS system needs to be made more robust to varying lighting conditions. The experiments (see Section 3 in Chapter 7) showed that the sensors are prone to give potential erroneous results in different environments. Testing with sensor processing algorithms should always try to vary some elements of the environment (e.g., wall types, lighting).
- (xi) There is a real need for real-time localization: fast algorithms for correcting the robot's position on the fly. The current localization procedure which correlates sonar measurements with an existing CAD map can take up to 30 sec (see Section 3.2 in Chapter 7).
- (xii) A global goal is projected onto the border of the local potential field. Dynamic mapping may cause the attraction of the projected global goal to be blocked within the confines of the local potential field. It is not clear how to guarantee that this does not occur with a simple algorithm, without requiring a reasoning agent to monitor and compensate for it (see Section 9.3 in Chapter 4 and Section 4 in Chapter 7).
- (xiii) Future work with SPOTT should include investigating learning within the scope of the SPOTT architecture. TR+ programs are very similar to AI decision trees and learning TR+ control programs and revising existing control programs would be a logical extension, especially when new environments are being explored. Another area to investigate is the learning of new conditions and actions.

The research issues discovered as part of experimenting with the SPOTT architecture are particular to the environment (i.e., office and laboratory) in which SPOTT was tested. Experiments (see Section 5 in Chapter 7) showed that many assumptions are implicitly made about the sensor processing algorithms when all experiments are confined to the same setting. Different settings must be tested to make the system as general and robust as possible.



#### 4. Summary

This thesis has proposed a behavioral-based robot control architecture called SPOTT which is distinguished from other behavioral architectures by its ability to guarantee task completion for a variety of tasks. The modularity and flexibility of the architecture permits the inclusion of a reasoning agent to advise SPOTT when it reaches its limitations. SPOTT has been actually implemented using a distributed architecture consisting of a collection of existing heterogeneous workstations using the message-passing software package called PVM. SPOTT has also been interfaced to a reasoning agent called COCOLOG. Preliminary investigation with the SPOTT architecture has provided a proof that the conceptual ideas that contribute to SPOTT's design can actually control a robot. Some engineering issues are currently being addressed to permit the robot to move at acceptable speeds (i.e., the average speed of a walking human, approximately 1 m/sec).

## REFERENCES

---

- Agre, Phillip E., & Chapman, David. 1987. Pengi: An Implementation of a Theory of Activity. *Pages 268-272 of: Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence*. AAAI.
- Aho, Alfred V., Hopcroft, John E., & Ullman, Jeffrey D. 1983. *Data Structures and Algorithms*. Addison-Wesley Publishing Co.
- Ames, William F. 1992. *Numerical Methods for Partial Differential Equations*. Academic Press Inc.
- Anderson, T.L., & Donath, M. 1991. Animal behavior as a paradigm for developing robot autonomy. *Pages 145-168 of: Designing Autonomous Agents*. MIT Press.
- Anderson, Tracy L., & Donath, Max. 1988. Synthesis of Reflexive Behaviors for a Mobile Robot Based on Stimulus-Response Paradigm. *Pages 370-382 of: Proceedings of the 1988 SPIE Conference on Mobile Robots*. SPIE.
- Arabe, Jose Nagib Cotrim, Beguelin, Adam, Lowekamp, Bruce, Seligman, Erik, Starkey, Mike, & Stephan, Peter. 1995 (April). *Dome: Parallel programming in a heterogeneous multi-user environment*. Tech. rept. Technical Report CMU-CS-95-137. School of Computer Science, Carnegie Mellon University, Pittsburg, PA.
- Arbib, Michael A., & House, Donald H. 1987. Depth and Detours: An Essay on Visually Guided Behavior. *Pages 129-163 of: Vision, Brain and Cooperative Computation*. MIT Press.
- Arkin, Ronald C. 1989. Motor Schema - Based Mobile Robot Navigation. *The International Journal of Robotics Research*, 8(4), 92-112.
- Arkin, Ronald C. 1990a. The Impact of Cybernetics on the Design of a Mobile Robot System: A Case Study. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(6), 1245-1257.
- Arkin, Ronald C. 1990b. Integrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation. *Robotics and Autonomous Systems*, 6, 105-122.
- Arkin, Ronald C. 1993. Behavior-based Robot Navigation for Extended Domains. *Journal of Adaptive Behavior*, 1(2), 75-99.
- Axler, S., Bourdon, P., & Ramey, W. 1991. *Harmonic Function Theory, vol. 137 of Graduate Texts in Mathematics*. New York: Springer-Verlag.
- Beguelin, A., Dongarra, J., Geist, A., & Sunderam, V. 1993. Visualization and Debugging in a Heterogeneous Environment. *IEEE Computer*, 26(6), 88-95.
- Benson, Scott, & Nilsson, Nils J. 1995. Reacting, Planning, and Learning in an Autonomous Agent. *Machine Learning* 14.
- Biederman, I. 1987. Recognition-by-components: A theory of human image understanding. *Psychological Review*, 94(2), 115-147.

- Birman, Kenneth, & Marzullo, Keith. 1989. Isis and the META project. *Sun Technology*, Summer, 90-104.
- Blais, Francois, Rioux, Marc, & Domey, Jacques. 1991. Optical Range Image Acquisition for the Navigation of a Mobile Robot. *Pages 2574-2586 of: Proceedings 1991 IEEE International Conference on Robotics and Automation*.
- Bolduc, Marc. 1996 (July). *The QUADRIS Sensor*. Tech. rept. CIM-TR-96-?? McGill Research Centre for Intelligent Machines, McGill University, Montreal, PQ, Canada.
- Borenstein, Johann, Everett, H.R., & Feng, Liquiang. 1996. *Navigating Mobile Robots: Systems and Techniques*. Wellesley, Massachusetts: ISBN 1-56881-058-X, A.K. Peters, Ltd.
- Bou-Ghannam, Akram. 1992. Controlling Reactive Behavior with Consistent World Modeling and Reasoning. *Pages 701-712 of: SPIE Vol. 1708 Applications of Artificial Intelligence X: Machine Vision and Robotics*.
- Bozma, O., & Kuc, R. 1991. Building a Sonar Map in a Specular Environment Using a Single Mobile Sensor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(12), 1260-1269.
- Brooks, Rodney A. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Transactions on Robotics and Automation*, 2(4), 14-23.
- Brooks, Rodney A. 1990 (April). *The Behavior Language: User's Guide*. Tech. rept. AI memo 1227. Massachusetts Institute of Technology, Cambridge, MA.
- Brooks, Rodney A. 1991. Challenges for Complete Creature Architectures. *Pages 434-443 of: From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*. Bradford Book, MIT Press.
- Bui, Don. in preparation. *QUADRIS: A Range Sensor for Navigation and Landmark Recognition*. Masters thesis, McGill University, Dept. of Electrical Engineering.
- Caines, P.E., & Wang, S. 1995. COCOLOG: A Conditional Observer and Controller Logic for Finite Machines. *SIAM J. Cont. and Opt.*, 33(6), 1687-1715.
- Carriero, Nicholas, & Gelernter, David. 1989. How to write parallel programs: A guide to the perplexed. *ACM Computing Surveys*, September, 323-357.
- Carver, Norman, & Lesser, Victor. 1992. *The Evolution of Blackboard Control Architectures*. Tech. rept. 92-71. University of Massachusetts, Department of Computer Science, Amherst, MA.
- Choi, Wonyun, & Latombe, Jean-Claude. 1991. A Reactive Architecture for Planning and Executing Robot Motions with Incomplete Knowledge. In: *IEEE/RSI International Workshop on Intelligent Robots and Systems IROS'91*. IEEE.
- Connell, Jonathan H. 1990. *Minimalist Mobile Robotics*. Academic Press Inc.
- Connell, Jonathan H. 1992. SSS: A Hybrid Architecture Applied to Robot Navigation. *Pages 2719-2724 of: Proceedings from the 1992 IEEE International Conference on Robotics and Automation*.
- Connolly, Christopher I. 1994. *Harmonic Functions As A Basis For Motor Control And Planning*. Ph.D. thesis, University of Massachusetts, Amherst, Massachusetts.
- Connolly, Christopher I., & Burns, J. Brian. 1992. The Planning of Actions in the Basal Ganglia. In: *Artificial Intelligence Planning Systems: Proceedings of the First International Conference (AIPS92)*. AAAI.
- Connolly, Christopher I., & Grupen, Roderic A. 1993. On the Applications of Harmonic Functions to Robotics. *Journal of Robotic Systems*, 10(7), 931-946.
- Connolly, Christopher I., & Grupen, Roderic A. 1994 (June). *Nonholonomic Path Planning Using Harmonic Functions*. Tech. rept. UM-CS-1994-050. University of Massachusetts, Amherst, MA.

- Cox, Ingemar J. 1991. Blanche - An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle. *IEEE Transactions on Robotics and Automation*, 7(2), 193-204.
- Crowley, James L. 1985. Navigation for an Intelligent Mobile Robot. *IEEE Transactions on Robotics and Automation*, 1(1), 31-41.
- de Saint Vincent, A. Robert. 1986. A 3D Perception System for the Mobile Robot Hilare. *Pages 1105-1111 of: Proceedings from the 1986 IEEE International Conference on Robotics and Automation, Volume 2.*
- Denofsky, Murray Elias. 1976 (February). *How Near is Near?* Tech. rept. AI Memo No. 344. MIT AI Lab.
- Dickmanns, E.D., Mysliwetz, B., & Christians, T. 1990. An Integrated Spatio-Temporal Approach to Automatic Visual Guidance of Autonomous Vehicles. *IEEE Transactions on Systems, Man and Cybernetics*, 37(6), 1273-1284.
- Doyle, Peter G., & Snell, J. Laurie. 1984. *Random Walks and Electric Networks*. The Mathematical Association of America.
- Drumheller, M. 1987. Mobile Robot Localization Using Sonar. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(2), 325-331.
- Dudek, Gregory, & Jenkin, Michael. 1993 (Feb.). A Multi-Layer Distributed Development Environment for Mobile Robotics. *Pages 542-550 of: Proceedings of the International Conference on Intelligent Autonomous Systems (IAS-3).*
- Dudek, Gregory, Jenkin, Michael, Milios, Evangelos, & Wilkes, David. 1993 (December). *Reflections on modelling a sonar range sensor*. Tech. rept. CIM-2-9. McGill Research Centre for Intelligent Machines. McGill University, Montreal, PQ, Canada.
- Durrant-Whyte, H.F. 1988. *Integration, Coordinations, and Control of Multi-Sensor Robot Systems*. Boston: Kluwer.
- Elfes, Alberto. 1987. Sonar-Based Real-World Mapping and Navigation. *IEEE Transactions on Robotics and Automation*, 3(3), 249-265.
- Elfes, Alberto. 1989. Using Occupancy Grids for Mobile Robot Perception and Navigation. *IEEE Computer Magazine*, 22(6), 46-58.
- Evans, J., Krishnamurthy, B., Barrows, B., Skewis, T., & Lumelsky, V. 1992. Handling Real-World Motion Planning: A Hospital Transport Robot. *IEEE Control Systems*, February, 15-20.
- Everett, H.R. 1995. *Sensors for Mobile Robots: Theory and Applications*. ISBN 1-56881-048-2, A.K. Peters. Ltd.
- Ezzati, M. 1995 (September). *Fast Image Segmentation Using Stereo Vision*. Masters thesis, McGill University, Dept. of Electrical Engineering.
- Ferguson, Innes A. 1992. Touring Machines: Autonomous Agents with Attitudes. *Computer Magazine*, 25(5), 51-55.
- Ferrari, Adam, & Sunderam, V.S. 1995. TPVM: Distributed Concurrent Computing with Lightweight Processes. *Pages 211-218 of: IEEE Symposium on High Performance Distributed Computing.*
- Fikes, R., Hart, P., & Nilsson, N. 1972. Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, 3, 251-288.
- Firby, R. James. 1987. An Investigation into Reactive Planning in Complex Domains. *Pages 202-206 of: Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence*. AAAI.
- Flower, J., Kolawa, A., & Bharadway, S. 1991. The express way to distributed processing. *Supercomputing Review*, May, 54-55.

- Flynn, Anita M., & Brooks, Rodney A. 1988. MIT Mobile Robots - What's Next? *Pages 611-617 of: Proceedings 1988 International Conference on Robotics and Automation.*
- Gansner, E.R., Koutsoufios, E., North, S.C., & Vo, K.P. 1993. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3), 214-230.
- Gat, Erann. 1991a. ALFA: A Language for Programming Reactive Robotic Control Systems. *Pages 1116-1121 of: Proceedings from the 1991 IEEE International Conference on Robotics and Automation.*
- Gat, Erann. 1991b. Robust Low-computation Sensor-driven Control for Task- Directed Navigation. *Page 2484 of: Proceedings from the 1991 IEEE International Conference on Robotics and Automation.*
- Gat, Erann. 1992. Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots. *Proceedings of the AAAI92.*
- Gat, Erann, Desia, R., Ivlev, R., Loch, J., & Miller, D.P. 1994. Behavior Control for Robotic Exploration of Planetary Surfaces. *IEEE Transactions on Robotics and Automation*, 10(4), 490-503.
- Geist, Al, Beguelin, Adam, Dongarra, Jack, Jian, Weicheng, Manchek, Robert, & Sunderam, Vaidyu. 1993 (May). *PVM 3 User's Guide and Reference Manual*. Tech. rept. ORNL-TM-12187. Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, USA.
- Geist, Al, Beguelin, Adam, Dongarra, Jack, Jian, Weicheng, Manchek, Robert, & Sunderam, Vaidy. 1994. *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press.
- Georgeff, Michael P., & Lansky, Amy L. 1987. Reactive Reasoning and Planning. *Pages 677-682 of: Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence*. AAAI.
- Gropp, W., Lusk, E., & Skjellum, A. 1994. *Using MPI*. MIT Press.
- Hager, G.D. 1990. *Task-Directed Sensor Fusion and Planning*. Norwell, MA: Kluwer.
- Harmon, S. 1987a. Autonomous Vehicles. *Pages 39-45 of: Encyclopedia of Artificial Intelligence*. John Wiley and Sons.
- Harmon, S. 1987b. Robots, Mobile. *Pages 957-963 of: Encyclopedia of Artificial Intelligence*. John Wiley and Sons.
- Hartley, Ralph, & Pipitone, Frank. 1991. Experiments with the Subsumption Architecture. *Pages 1652-1658 of: Proceedings of the 1991 IEEE International Conference on Robotics and Automation*. IEEE.
- Hebert, Martial. 1989. Building and Navigating maps of road scenes using an active sensor. *Pages 1136-1142 of: Proceedings 1989 IEEE International Conference on Robotics and Automation*.
- Herskovits, Annette. 1985. Semantics and Pragmatics of Locative Expressions. *Cognitive Science*, 9, 341-378.
- Herskovits, Annette. 1988. Spatial Expressions and the Plasticity of Meaning. *Pages 271-297 of: Topics in Cognitive Linguistics*. Amsterdam/Philadelphia: John Benjamins Publishing Company.
- Hexmoor, Henry, & Kortenkamp, David. 1995. Issues on Building Software for Hardware Agents. *Knowledge Engineering Review*, 10(3), 301-304.
- Hodges, John C., & Whitten, Mary E. 1986. *Harbrace College Handbook*. 10 edn. New York: Harcourt Brace Jovanovich, Inc.
- Horn, B.K.P. 1977. Understanding Image Intensities. *Artificial Intelligence*, 8(2), 201-231.
- Hornbeck, Robert W. 1975. *Numerical Methods*. Quantum Publishers Inc.
- Horswill, Ian. 1995 (March 27-29). Taking (Computer) Architecture Seriously. *In: 1995 AAAI Spring*

*Symposium: Lessons Learned from Implemented Software Architectures for Physical Agents.*

Hwang, Yong K., & Ahuja, Narendra. 1992. Gross Motion Planning - A Survey. *ACM Computing Surveys*, 24(3), 220-291.

Iyengar, S. Sitharama, & Kashyap, Rangasami L. 1989. Autonomous Intelligent Machines. *IEEE Computer*, 22(6), 14-15.

Jackendoff, Ray. 1990. *Semantic Structures*. MIT Press.

Jarvis, R. 1983. A perspective on range finding techniques for computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2), 122-139.

Johnson-Laird, P.N. 1989. Cultural Cognition. *Pages 469-499 of: Foundations of Cognitive Science*. MIT Press.

Kadonoff, Mark, Benayad-Cherif, Faycal, Franklin, Austin, Maddox, James, Muller, Lon, & Moravec, Hans. 1988. Arbitration of Multiple Control Strategies for Mobile Robots. *Pages 90-98 of: Proceedings of the 1986 SPIE Conference on Mobile Robots*. SPIE.

Kaelbling, Leslie Pack. 1988. Goals as Parallel Program Specifications. *Pages 60-65 of: Proceedings of AAAI-88*. AAAI.

Kaelbling, Leslie Pack, & Rosenschein, Stanley J. 1990. Action and Planning in Embedded Agents. *Robotics and Autonomous Systems*, 6(1), 35-48.

Khatib, Oussama. 1986. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research*, 5(1), 90-98.

Koutsofios, Elefthereos, & Dobkin, David. 1991. Lefty: A two-view editor for technical pictures. *Pages 68-76 of: Graphics Interface 1991*.

Koutsofios, Elefthereos, & North, Stephen C. 1994 (May). Applications of Graph Visualization. *Pages 235-245 of: Proceedings of Graphics Interface 1994 Conference*.

Kriegman, David J., Triendl, Ernst, & Binford, Thomas O. 1989. Stereo Vision and Navigation in Buildings for Mobile Robots. *IEEE Transactions on Robotics and Automation*, 5(6), 792-803.

Krogh, B.H., & Thorpe, C.E. 1986. Integrated Path Planning and Dynamic Steering Control for Autonomous Vehicles. *Pages 1664-1669 of: Proceedings of the 1986 IEEE International Conference on Robotics and Automation*. IEEE Press.

Krotkov, Eric. 1989a. Mobile Robot Localization Using A Single Image. *Pages 978-983 of: Proceedings 1989 IEEE International Conference on Robotics and Automation*.

Krotkov, Eric Paul. 1989b. *Active Computer Vision by Cooperative Focus and Stereo*. Springer-Verlag.

Kweon, I., Kuno, Y., Watanabe, M., & Onoguchi, K. 1992. Behavior-Based Mobile Robot using Active Sensor Fusion. *Pages 1675-1682 of: Proceedings from the 1992 IEEE International Conference on Robotics and Automation*.

Landau, Barbara, & Jackendoff, Ray. 1993. What and Where in Spatial Language and Spatial Cognition. *Behavioral and Brain Sciences*, 16, 217-265.

Latombe, Jean-Claude. 1991. *Robot Motion Planning*. Kluwer Academic Publishers.

Leonard, John J., & Durrant-Whyte, Hugh F. 1991. Mobile Robot Localization by Tracking Geometric Beacons. *IEEE Transactions on Robotics and Automation*, 7(3), 376-382.

Longuet-Higgins, H.C. 1981. A Computer Algorithm for Reconstructing a Scene from two Projections. *Nature*, 293, 133-135.

- Lueth, T.C., Laengle, Th., Herzog, G., Stopp, E., & Rembold, U. 1994. KANTRA: Human-Machine Interaction for Intelligent Robots using Natural Language. *Pages 106-111 of: IEEE International Workshop on Robot and Human Communications.*
- Luo, Ren C., & Kay, Michael G. 1989. Multisensor Integration and Fusion in Intelligent Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5), 901-931.
- Lyons, Damian M. 1993. Representing and Analyzing Action Plans as Networks of Concurrent Processes. *IEEE Transactions on Robotics and Automation*, 9(3), 241-256.
- Mackenzie, Paul, & Dudek, Gregory. 1994. Precise Positioning Using Model-Based Maps. *Pages 11615-1621 of: IEEE International Conference on Robotics and Automation.* vol. 2. San Diego, CA: IEEE.
- Maes, Pattie. 1991. *Designing Autonomous Agents.* MIT Press.
- Manz, Allan, Liscano, Ramiro, & Green, David. 1991. A Comparison of Realtime Obstacle Avoidance Methods for Mobile Robots. In: *Second International Symposium on Experimental Robotics, Preprints.* LAAS/CNRS, Toulouse (France).
- Marr, D., & Poggio, T. 1976. Cooperative computation of stereo disparity. *Science*, 194, 283-287.
- Mataric, Maja J. 1992. Integration of Representation into Goal-Driven Behavior-Based Robots. *IEEE Transactions on Robotics and Automation*, 8(3), 304-312.
- Matthies, Larry, Kanade, Takeo, & Szeliski, Richard. 1989. Kalman Filter-based Algorithms for Estimating Depth from Image Sequences. *International Journal of Computer Vision*, 3, 209-236.
- McCann, G.D., & Wilts, C.H. 1949. Application of Electric-Analog Computers to Heat-Transfer and Fluid-Flow Problems. *Journal of Applied Mechanics*, 16(3), 247-258.
- McFarland, David. 1989. *Problems of Animal Behavior.* Longman Scientific and Technical, John Wiley and Sons.
- Miller, George A., & Johnson-Laird, Philip N. 1976. *Language and Perception.* Harvard University Press.
- Moravec, Hans P. 1983. The Stanford Cart and the CMU Rover. *Proceedings of the IEEE*, 71(7), 872-884.
- Moravec, Hans P. 1988. Sensor Fusion in Certainty Grids for Mobile Robots. *AI Magazine*, 9(2), 61-74.
- Mouaddib, Abdel Illah, & Zilberstein, Shlomo. 1995. Knowledge-Based Anytime Computation. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1(August 20-25), 775-781.
- Moutarlier, P., & Chatila, R. 1990. Incremental Environment Modelling by a Mobile Robot from Noisy Data. *Pages 327-346 of: Experimental Robotics 1.* Springer-Verlag.
- Murphy, Robin R. 1996. Biological and Cognitive Foundations of Intelligent Sensor Fusion. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 26(1), 42-51.
- Musliner, David J., Hendler, James A., & Agrawala, Ashok K. 1994 (June). *The Challenges of Real-Time AI.* Tech. rept. CS-TR-3290, UMIACS-TR-94-69. University of Maryland Technical Report, Maryland, USA.
- Newell, Allen. 1990. *Unified Theories of Cognition.* Harvard University Press.
- Nilsson, Nils J. 1992 (January). *Toward Agent Programs With Circuit Semantics.* Tech. rept. STAN-CS-92-1412. Department of Computer Science, Stanford University, Stanford, California 94305.
- Nilsson, Nils J. 1994. Teleo-Reactive Programs for Agent Control. *Journal of Artificial Intelligence Research*, 1, 139-158.
- Nilsson, Nils J. 1995. Eye on the Prize. *Artificial Intelligence Magazine*, July.
- Nilsson, N.J. 1969. A Mobile Automaton: An Application of Artificial Intelligence Techniques. *Pages*

- 509-520 of: *Proceedings of the First International Joint Conference on Artificial Intelligence, Washington, DC.*
- Noreils, Fabrice R., & Prajoux, Roland. 1991. From Planning to Execution Monitoring Control for Indoor Mobile Robots. *Pages 1510-1517 of: Proceedings from the 1991 IEEE International Conference on Robotics and Automation.*
- North, Stephen C. 1993. Drawing Ranked Digraphs with Recursive Clusters. (*submitted*). Abstract presented at Proc. ALCOM International Workshop on Graph Drawing and Topological Graph Algorithms. Paris, 1993.
- Paul, C.J., Acharya, A., Black, B., & Strosnider, J.K. 1991. Reducing Problem-Solving Variance to Improve Predictability. *Communications of the ACM*, 34(8), 81-93.
- Payton, David W. 1986. An Architecture for Reflexive Autonomous Vehicle Control. *Pages 1838-1845 of: Proceedings from the 1986 IEEE International Conference on Robotics and Automation.*
- Prazdny, K. 1980. Egomotion and relative depth map from Optical Flow. *Biological Cybernetics*, 26, 87-102.
- Rencken, W.D. 1994 (September). Autonomous Sonar Navigation in Indoor, Unknown, and Unstructured Environments. *Pages 127-134 of: 1994 International Conference on Intelligent Robots (IROS'94).*
- Rimon, Elon, & Koditschek, Daniel E. 1990. Exact Robot Navigation in Geometrically Complicated but Topologically Simple Spaces.
- Robinson, David Lee, & Petersen, Steven E. 1992. The pulvinar and visual salience. *Trends in Neuroscience*, 15(4), 127-132.
- Sabersky, R.H., Acosta, A.J., & Hauptmann, E.G. 1971. *Fluid Flow*. 2nd edn. MacMillan Publishing Company.
- Schiele, B., & Crowley, J. 1994. A Comparison of Position Estimation Techniques Using Occupancy Grids. *Pages 163-171 of: Proceedings of the IEEE Conference on Robotics and Automation*. San Diego, CA: IEEE.
- Simmons, Reid G. 1994. Structure Control for Autonomous Robots. *IEEE Transactions on Robotics and Automation*, 10(1), 34-43.
- Simon, Daniel, Espiau, Bernard, Castillo, Eduardo, & Kapellos, Konstantinos. 1993. Computer-Aided Design of a Generic Robot Controller Handling Reactivity and Real-Time Control Issues. *IEEE Transactions on Control Systems Technology*, 1(4), 213-229.
- Smith, M.H., Sobek, R.P., Coles, L.S., Hodges, D.A., & Roby, A.M. 1975. The System Design of JASON, A Computer Controlled Mobile Robot. *Pages 72-75 of: Proceedings of the International Conference on Cybernetics and Society IEEE, New York.*
- Stan, Mircea R., Burleson, Wayne P., Connolly, Christopher I., & Grupen, Roderic A. 1994. Analog Path Computation Using VLSI Relaxation Networks. *Journal of VLSI Signal Processing*, 8(1).
- Sunderam, V.S., Geist, G.A., Dongarra, J., & Manchek, R. 1993. The PVM Concurrent Computing System: Evolution, Experiences, and Trends. *Journal of Parallel Computing*, 20(4), 531-546.
- Talluri, R., & Aggarwal, J. 1993. Position Estimation Techniques for an Autonomous Mobile Robot - A Review. *Chap. 4.4, pages 769-801 of: Handbook of Pattern Recognition and Computer Vision*. Singapore: World Scientific.
- Tarassenko, L., & Blake, A. 1991. Analogue computation of collision-free paths. *Pages 540-545 of: Proceedings of the 1991 IEEE International Conference on Robotics and Automation*. IEEE.
- Thorpe, Charles, Hebert, Martial H., Kanade, Takeo, & Shafer, Steven A. 1988. Vision and Navigation



- for the Carnegie-Mellon Navlab. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3), 362-373.
- Tinbergen, Nikolaas. 1951. *The Study of Instinct*. Clarendon Press, Oxford.
- Topol, Brad, Stasko, John T., & Sunderam, Vaidy. 1994 (October). *Integrating Visualization Support into Distributed Computing Systems*. Tech. rept. Technical Report GIT-GVU-94/38. Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA.
- van de Vooren, A.I., & Vliegthart, A.C. 1967. On the 9-Point Difference Formula for Laplace's Equation. *Journal of Engineering Mathematics*, 1(3), 187-202.
- Vandeloise, Claude. 1991. *Spatial Prepositions: A Case Study from French*. The University of Chicago Press. Translated by R.K. Bosch.
- Watanabe, M., Onoguchi, K., Kweon, I., & Kuno, Y. 1992. Architecture of Behavior-based Mobile Robot in Dynamic Environment. Pages 2711-2718 of: *Proceedings from the 1992 IEEE International Conference on Robotics and Automation*.
- Waxman, Allen M., Kushner, Todd R., Liang, Eli, & Siddalingariah, Thrakesh. 1987. A Visual Navigation System for Autonomous Land Vehicles. *IEEE Transactions on Robotics and Automation*. 3(2), 124-141.
- Weiβ, G., Wetzler, C., & Puttkamer, E. 1994 (September). Keeping Track of Position and Orientation of Moving Indoor Systems by Correlation of Ranger-Finder Scans. Pages 595-601 of: *1994 International Conference on Intelligent Robots (IROS'94)*.
- Witkin, Andrew P. 1981. Recovering Surface Shape and Orientation from Texture. *Artificial Intelligence*. 17(1), 17-45.
- Wu, Kenong. 1996. *Computing Parametric Geon Descriptions of 3D Multi-part Objects*. Ph.D. thesis. McGill University, Dept. of Electrical Engineering.
- Wu, Keonong, & Levine, Martin D. 1993 (September). *3-D Object Representation Using Parametric Geons*. Tech. rept. TR-CIM-93-13. McGill Research Centre for Intelligent Machines, McGill University, Montreal, PQ, Canada.
- Yuta, Shinichi, Suzuki, Shooji, & Iida, Shigeki. 1991. Implementation of a small size experimental self-contained autonomous robot. In: *Second International Symposium on Experimental Robotics, Preprints*. LAAS/CNRS, Toulouse (France).
- Zadeh, L.A. 1974. *A fuzzy-algorithm approach to the definition of complex or imprecise concepts*. Tech. rept. Memo No. ERL-M474. Electronic Research Laboratory, University of California, Berkeley.
- Zhau, T.C., & Overmans, Mark. 1995. *Forms Library: A Graphical User Interface Toolkit*. Dept. of Physics. University of Wisconsin-Milwaukee, Milwaukee WI 53201, USA.

## APPENDIX A

---

### Harmonic Function Computability

The harmonic function is the solution to Laplace's equation and a gradient descent on this function is used as a path planning strategy (See Chapter 4). Recall that the potential function is computed in the free space, which is the space void of all obstacle and goal models. The extent of the potential function, as well as the boundaries of the obstacle and goal models define where the boundary conditions are applied.

For 2D, Laplace's equation is given by the following:

$$(A.A.1) \quad \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = 0$$

The solution is obtained by assuming  $U = XY$ , where  $X$  depends only on  $x$  and  $Y$  depends only on  $y$ . Using the separation of variables method results in the following:

$$(A.A.2) \quad \begin{aligned} X''Y + XY'' &= 0 \\ \frac{X''}{X} &= -\frac{Y''}{Y} \end{aligned}$$

The second equation in Equation A.A.2 is obtained by dividing the first equation by  $XY$ . If each side of the second equation is set equal to  $-\lambda^2$ , then the second equation in Equation A.A.2 can be rewritten as the following two equations:

$$(A.A.3) \quad \begin{aligned} X'' + \lambda^2 X &= 0 \\ Y'' - \lambda^2 Y &= 0 \end{aligned}$$

which can be equivalently presented as follows:

$$(A.A.4) \quad \begin{aligned} X &= c_1 \cos \lambda x + c_2 \sin \lambda x \\ Y &= c_3 e^{\lambda y} + c_4 e^{-\lambda y} \end{aligned}$$

Assuming that  $\lambda > 0$ , this solution becomes unbounded as  $y \rightarrow \infty$ , which violates the bounding condition (see the first paragraph in Sections 1 and 2). To avoid catastrophe,  $c_3 = 0$  is chosen. This causes the solution to take on the following form:

$$(A.A.5) \quad U(x, y) = e^{-\lambda y} (A \cos \lambda x + B \sin \lambda x)$$

where  $A = c_1 c_4$  and  $B = c_2 c_4$ .

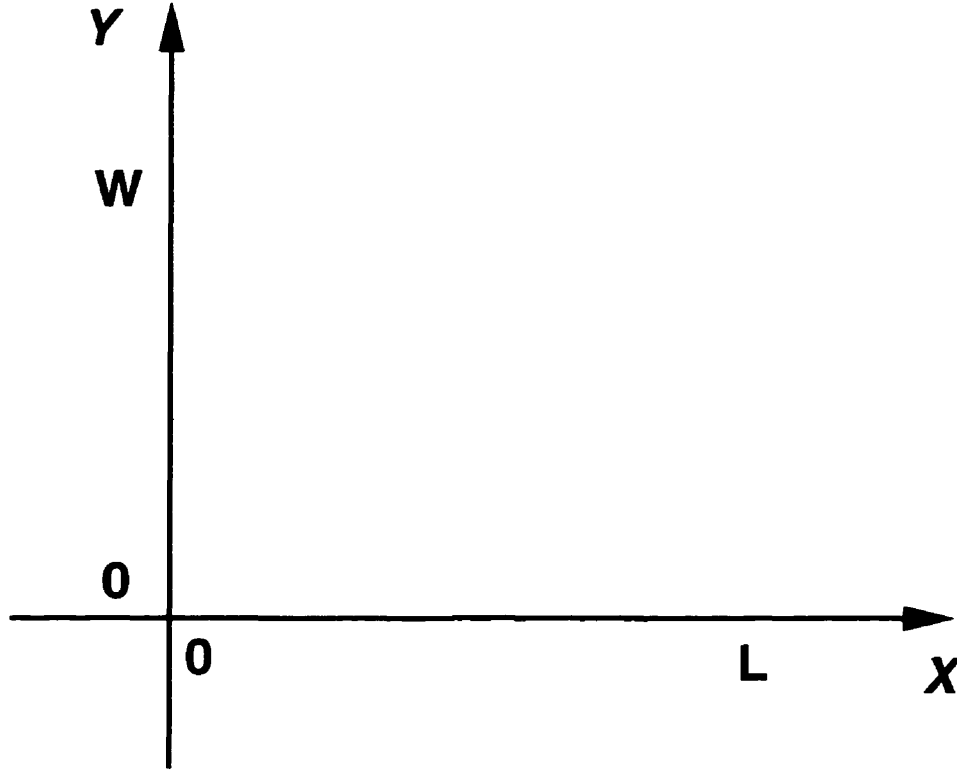


FIGURE A.1. **Potential Function Extent:** The derivations for Neumann and Dirichlet boundary conditions are formulated in the shaded areas, given the boundaries shown above.

### 1. Neumann Boundary

Neumann boundary conditions are given by  $\frac{\partial U(x, W)}{\partial y} = 0$ ,  $\frac{\partial U(x, 0)}{\partial y} = 0$ ,  $U(0, y) = U_M$ , and  $U(L, y) = U_0$ . In order to simplify the computations, the constants  $U_M$  and  $U_0$  are set as follows:  $U_M = 1$  and  $U_0 = 0$ .  $U(x, y)$  is also bounded:  $|U(x, y)| < U_m = 1$ .

Applying the boundary condition  $\frac{\partial U(x, 0)}{\partial y} = 0$  results in solutions of the following form:

$$(A.A.6) \quad \begin{aligned} X(x) &= A_1 e^{-kx} \\ Y(y) &= A_1 \cos ky \end{aligned}$$

The following results when combining the two solutions given in Equation A.A.6:

$$(A.A.7) \quad U(x, y) = C_n e^{-kx} \cos ky$$

Applying the other boundary condition  $\frac{\partial U(x, W)}{\partial y} = 0$  produces  $kW = n\pi$ . Therefore the solution is simply  $U(x, y) = \frac{L-x}{L}$ . The gradient of this function is  $\nabla\phi = -\frac{1}{L}$ , which is a constant.

When using Neumann boundary conditions, there is no limit imposed on the size of the potential field, as  $2^{32} - 1$  could be the theoretical dimension of one of the sides for the potential field. The real constraining factor is that for this size, the potential function would require many iterations to compute and would be practically unfeasible.

## 2. Dirichlet Boundary

Dirichlet boundary conditions for the bounds specified in Figure A.1 are given by  $U(x, 0) = U_m$ ,  $U(x, W) = U_M$ ,  $U(0, y) = U_M$ , and  $U(L, y) = U_0$ . In order to simplify the computations, the constants  $U_M$  and  $U_0$  are set as follows:  $U_M = 1$  and  $U_0 = 0$ .  $U(x, y)$  is also bounded:  $|U(x, y)| < U_m = 1$ .

Applying the boundary condition  $U(x, 0) = 1$  results in  $Y(y) = B_1 \cos ky$ . Therefore, the solution takes on the following form:

$$(A.A.8) \quad U(x, y) = C e^{-kx} \cos ky$$

Applying the boundary condition  $U(x, W) = 1$  results in the following:

$$(A.A.9) \quad k = \frac{n\pi}{W}, n = 1, 2, 3, \dots$$

and the solution takes the following form:

$$(A.A.10) \quad U(x, y) = C_n e^{-\frac{n\pi x}{W}} \cos \frac{n\pi}{W} y$$

Using the property of superposition and the boundary condition  $U(0, y) = U_M$  results in the following:

$$(A.A.11) \quad U(0, y) = \sum_1^\infty U_n(0, y) = \sum C_n \cos \frac{n\pi}{W} y = 1$$

This reduces to  $C_n = \frac{2}{n\pi}$  if  $n$  is odd, and  $C_n = 0$  if  $n$  is even. Substituting Equations A.A.8 and A.A.9 into A.A.10 produces the following:

$$\begin{aligned}
 (A.A.12) \quad U(x, y) &= \sum_{n=1}^{\infty} C_n e^{-\frac{n\pi x}{W}} \cos \frac{n\pi}{W} y \\
 &= \frac{2}{\pi} \sum_{m=1}^{\infty} \frac{1}{2m+1} e^{-\frac{n\pi x}{W}} \cos \frac{n\pi}{W} y \\
 &= \frac{4}{\pi} \sum_{n=1}^{\infty} \frac{1}{2m+1} e^{-\frac{(2m+1)\pi x}{2W}} \cos \frac{(2m+1)\pi y}{2W}
 \end{aligned}$$

Therefore,  $|\nabla\phi|$  decays as fast as  $\exp(-\frac{\pi x}{2W})$ , the worst case being  $\exp(-\frac{L\pi}{2W})$ . This means that approximately  $9.06 \frac{L}{W}$  bits of resolution are required to represent the rapidly decaying potential function. This also implies that for a 32 bit machine (i.e., using double precision), only corridors that have a length to width ratio of less than approximately 7.1 can be computed<sup>1</sup>.

Dirichlet boundary conditions define a limiting bound on the size of the potential function, whereas, Neumann boundary conditions do not. Dirichlet boundary conditions (see Figure 4.6) on the obstacles produce trajectories that smoothly go around the obstacle. The separation from the obstacle is determined by the configuration of obstacles (e.g., when there are two obstacles and the trajectory goes between them, the trajectory is centered). On the other hand, Neumann boundary conditions (see Figure 4.5) produce trajectories which hug the obstacles. Dirichlet boundary conditions have been chosen as boundary conditions to apply to the obstacle boundaries for this very reason. The cost for this is that for indoor environments, where narrow corridors (e.g., hallways) are prevalent, the potential field is local in scope. The size of the potential function is constrained by the length to width ratio (i.e., less than 7.1 for a 32 bit machine) of the hallways. The other factor considered is the computation time of the potential function, which is exponential in relation to the number of grid elements.

<sup>1</sup>This is based on using double precision (i.e., 64 bits) to represent numbers between 0 and 1. This is found by setting  $\exp(-\frac{L\pi}{2W})$  equal to  $2^{-b}$ , where  $b$  is the number of bits.

## APPENDIX B

---

### TR+ Conditions and Actions Implemented as Part of SPOTT

#### 1. Conditions

A TR+ condition is a Boolean expression which usually initiates a computation process which may take inputs or pass outputs through variables. The condition is evaluated as a UNIX process.

The condition names are all lower case, and are descriptive of their function. There are two parameters which are specified by the user with respect to a particular condition:

- (i) The type of architecture that the condition process executes on (or even the specific workstation) may be specified. In the default case, PVM chooses the workstation that executes the condition process.
- (ii) The only other parameter a user may specify about a particular executing condition process is the frequency at which it is computed.

The conditions implemented as part of SPOTT and a short description of what they compute are as follows:

- **all room targets explored:** This condition returns a TRUE value if there are no more room targets (i.e., goals) to enter. The condition “**find a new room target**” updates a shared variable containing a list of the rooms (i.e., a type of node in the abstract graph) already visited.
- **all targets within room explored:** This condition returns a TRUE value if there are no more locations to visit within a particular room. Typically, a target within a room is determined by finding the largest open space.
- **bumper has collided:** This condition returns TRUE if the bumper sensor has been triggered. Variables associated with this condition return the position and orientation of the bumper collision and a position to move robot to so as to move away from the collision site. The position to move the robot to is usually oriented

in the direction of  $180^\circ$  away from the collision, and a range of one robot's diameter away from the collision location.

- **current robot position is:** This condition always returns TRUE and invokes the algorithm responsible for localizing the robot based on sonar data. This algorithm correlates a sonar scan with the existing CAD map to get an estimate of the robot's pose (Mackenzie & Dudek, 1994).
- **door not open:** This condition returns TRUE if the nearest doorway which provides access into the current room is blocked (e.g., door is closed).
- **find a new room:** This condition always returns TRUE, except when there is no new room to visit. The variable associated with the condition returns a new intermediate goal (i.e., room) to visit. This condition keeps track of all the rooms and continually selects a new room. The set of rooms is determined from scanning the abstract graph.
- **find a new room target:** This condition always returns TRUE. The variable associated with this condition returns a new target (i.e., goal) based on the largest open space in the current room (see "**all room targets explored**").
- **find a new target within room:** This condition always returns TRUE. The variable associated with this condition returns intermediate goals within the room. The intermediate goals are usually found by finding the largest open space from the robot's current position.
- **find new direction target:** This condition always returns TRUE. It returns a target (i.e., goal) based on the direction specified in the task command.
- **find new random target:** This condition always returns TRUE. It returns a random target for wandering. A random number generator is called which is mapped into a location in a local region based around the robot's current position.
- **in hallway:** This condition returns TRUE if the robot is in a hallway, else it returns FALSE.
- **in room:** This condition returns TRUE if the robot is in a room, else it returns FALSE.
- **infrared shows objects very close:** This condition returns TRUE if the infrared sensors have detected an object within 60 cm of the robot. Variables associated with this condition return the position and orientation of the sensed environmental feature with respect to the robot's current position.

- **is danger target computed:** This condition always returns TRUE. This condition returns a goal position (i.e., in its output variable) which is out of harms way of another dangerous position (i.e., in its input variable).
- **is intermediate target reached:** This condition returns TRUE if the intermediate goal set by one of the TR+ conditions has been reached. The intermediate goal is a global variable.
- **is intermediate target set:** This condition returns TRUE if the intermediate goal has been set by a TR+ condition. The intermediate goal is a global variable.
- **is task target location known:** This condition returns TRUE if the spatial location of the target specified by the task command is known. The task command is a global variable.
- **is task target reached:** This condition returns TRUE if the target (i.e., goal) specified by the task command has been reached.
- **is the variable false:** This conditions checks the Boolean value of an input variable. If the input variable is FALSE, then the condition returns TRUE, and visa versa.
- **moving the robot failed:** This condition returns TRUE if there has been some problem found in moving the robot. The state of the robot is returned from the low level robot control software (i.e., *robodaemon*).
- **new object positions are:** This condition returns TRUE when new sonar data has been sensed. This condition initiates a sonar scan and the clustering of the sonar points into line segments (Mackenzie & Dudek, 1994). The output variable contains a list of line segments found from the sonar scan.
- **object labels are:** This condition returns TRUE when new QUADRIS data has been sensed. The output variable contains a list of line segments found from the range sensing scan (Bui, in preparation). The set of line segments may also have a label attached to them identifying them as a particular type of object (e.g., wall, door).
- **partial map available:** This condition returns TRUE when a partial a priori map is available (e.g., architectural CAD map).
- **robot in a room:** This condition returns TRUE if the robot is in the room specified by an input variable.
- **room not explored:** This condition returns TRUE if the room specified by an input variable has not yet been visited during the current execution of the task.



- **task has no destination just direction:** This condition returns TRUE if the task command has no destination specified, and only a direction element is specified.
- **task has no destination no direction:** This condition returns TRUE if the task command has no destination or direction element specified.

## 2. Actions

Actions either modify elements in the potential field or drive the robot actuators directly. The elements which can be modified in the potential field include adding obstacle or goal information, or correcting the current estimate of the robot's position. The names of the actions are all in lower case, and there has been an attempt to make the action names into sentence form. The actions implemented as part of SPOTT and a short description of what they perform are as follows:

- **remove all intermediate targets:** This action removes all the set intermediate targets. The intermediate targets are stored as part of a global variable.
- **remove intermediate target:** This action removes a particular intermediate target as specified in an input variable.
- **set intermediate target to:** This action sets the intermediate target to what is indicated in the input variable.
- **set label positions to:** This action updates the potential field with a list of mapping features (i.e., line segments). The line segments may have labels (e.g., door, wall) attached to them, as provided by the QUADRIS sensor processing algorithm (Bui, in preparation). This action typically operates with the QUADRIS sensing condition "**object labels are**".
- **set object positions to:** This action updates the potential field with a list of mapping features (i.e., line segments) (Mackenzie & Dudek, 1994). This action typically operates with the sonar sensing condition "**set object positions to**".
- **set ptu to hallway mode:** The PTU's for the QUADRIS system have different scanning routines depending on the context. This action sets the scanning routine to the hallway mode.
- **set ptu to room mode:** This action sets the scanning routine for the QUADRIS PTU to the hallway mode.
- **set real world coord to:** This action changes the definition for the global frame of reference for the Cartesian coordinate system.

- **set robot position to:** This action updates the current estimate of the robot's position (Mackenzie & Dudek, 1994). It is typically used in conjunction with the condition "**current robot position is**".
- **set scale to:** This action changes the scale of the highest resolution potential field. Typically, the resolutions of the other potential fields (i.e., if a multi-resolution scheme is used) are kept in direct proportion to the highest resolution one (i.e., specified as an input parameter).
- **set variable to false:** This action sets the value of its input variable to FALSE.
- **set variable to true:** This action sets the value of its input variable to TRUE.
- **turn the robot ctrl off:** This action stops the *robot actuator* process from performing steepest gradient descent on the potential function.
- **turn the robot ctrl on:** This action undoes "**turn the robot ctrl off**" and puts the *robot actuator* process back in its normal operating state (i.e., the robot is moved based on the steepest gradient descent found in potential function at the robot's current estimated position).

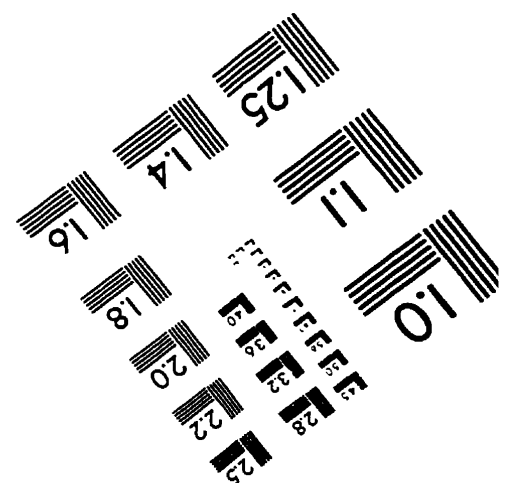
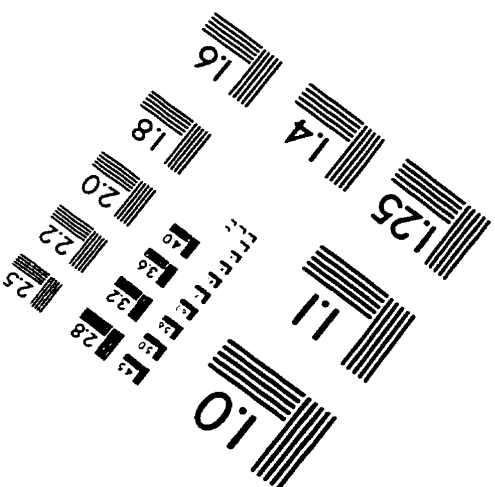
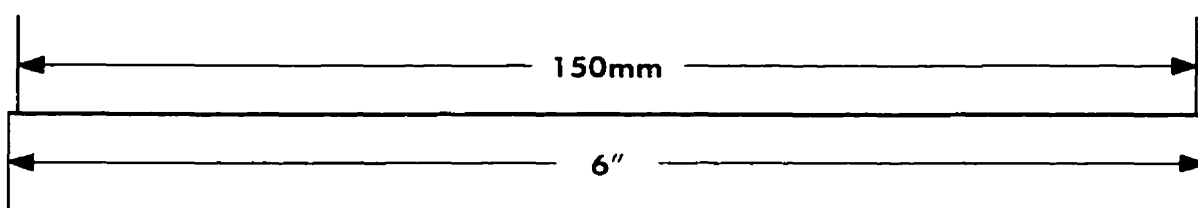
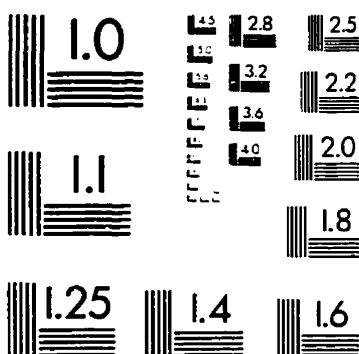
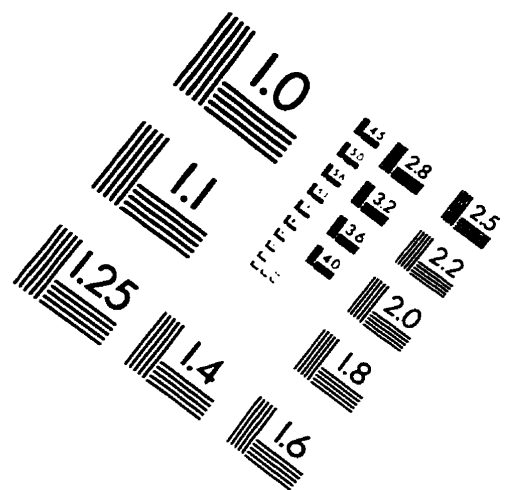
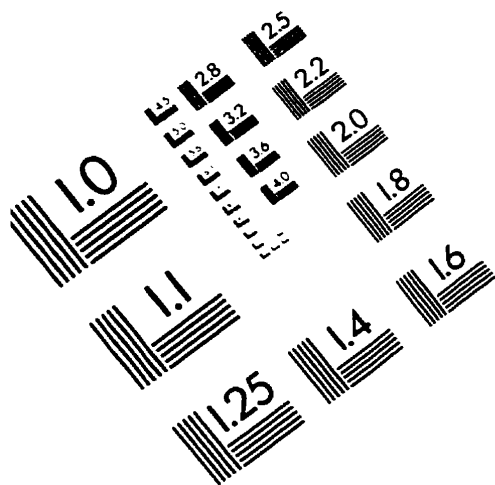
### 3. Variables

Variables are used to pass information between condition, actions, and the main and subroutine TR+ programs. They have to be predefined. They are created for every instance in every TR+ sub-routine they pop up in. The variable names are specified by using only capital letters. The variables used as part of SPOTT's current implementation include the following:

- **COORD\_REF:** This variable specifies the coordinate reference for the internal Cartesian map.
- **INTERMEDIATE\_TARGET :** This variable is used to store the intermediate target (i.e., goal position).
- **LABEL:** This variable is used to store a list of objects (i.e., line segments) which may or may not have a label (e.g., door, wall) associated with them.
- **OBJECT\_POSITIONS:** This variable is used to store a list of objects (i.e., line segments) which are typically returned from the sonar processing.
- **ROBOT\_POSITION:** This variable is used to store the robot's position and pose.
- **ROOM :** This variable stores a room number.
- **ROOM\_STATUS:** This variable stores a Boolean value which indicates whether a particular room has been searched or not.

- **SCALE:** This variable stores the scale (i.e., resolution) of the highest resolution potential function.
- **STATUS:** This is a Boolean variable, usually indicating whether something has gone wrong (e.g., robot failure). This variable is normally TRUE, but is FALSE if something has gone wrong.

# IMAGE EVALUATION TEST TARGET (QA-3)



APPLIED IMAGE, Inc  
1653 East Main Street  
Rochester, NY 14609 USA  
Phone: 716/482-0300  
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved